Acta Univ. Sapientiae, Informatica **10,** 2 (2018) 183–217

DOI: 10.2478/ausi-2018-0010

Sampling k-partite graphs with a given degree sequence

Koko K. Kayibi
Department of Mathematics, University of Bristol, United Kingdom
email: kokokayibi@yahoo.co.uk

U. Samee
Department of Mathematics, Islamia College for Science and Commerce, Srinagar, India
email: drumatulsamee@gmail.com

Shariefuddin Pirzada
University of Kashmir, Srinagar, India
email: pirzadasd@kashmiruniversity.ac.in

Muhammad Ali Khan
Department of Mathematics and Computer Science, University of Lethbridge, Canada
email: ma.khan@uleth.ca

Abstract. The authors in the paper [15] presented an algorithm that generates uniformly all the bipartite realizations and the other algorithm that generates uniformly all the simple bipartite realizations whenever A is a bipartite degree sequence of a simple graph. The running time of both algorithms is $\mathcal{O}(m)$, where $m = \frac{1}{2}\sum_{i=1}^{n} a_i$. Let $A = (A_1 : A_2 : \ldots : A_k)$ be a k-partite degree sequence of a simple graph, where A_i has n_i entries such that $\sum n_i = n$. In the present article, we give a generalized algorithm that generates uniformly all the k-partite realizations of A and another algorithm that generates uniformly all the simple k-partite realizations of A. The running time of both algorithms is $\mathcal{O}(m)$, where $m = \frac{1}{2}\sum_{i=1}^{n} a_i$.

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C07, 65C05
Key words and phrases: Degree sequence, contraction of a degree sequence, degree sequence bipartition, contraction of a graph, deletion of a graph, ecological occurrence matrix

1 Introduction

A k-*partite* graph G is a graph whose vertex set, denoted by $V(G)$, can be partitioned into k parts, $V_1(G), V_2(G), ..., V_k(G)$, such that two vertices in the same part are not adjacent. That is, if $E(G)$ denotes the edge set of G and $e = (v_i, v_j) \in E(G)$, then $v_i \in V_s(G)$ and $v_j \in V_r(G)$ such that $s \neq r$. An edge $e \in E(G)$ is said to be a *multiple* edge if there is another edge f incident to the same vertices. Following Matroid Theory terminology, we say that e and f are *parallel*. A *simple* k-partite graph is a k-partite graph with no multiple edges and loops. The *degree* of a vertex v_i, denoted by a_i, is defined as the number of edges incident to v_i with a loop contributing twice to the degree of v_i. The degree sequence of a graph G is formed by listing the degrees of vertices of G. If $A = (a_1, a_2, ..., a_n)$ is a sequence of integers and G is a k-partite graph that has A as its degree sequence, we say that G is a *realization* of A, and such a sequence of integers is called a k-*partite degree sequence*. Thus entries of A can be partitioned as $A_1, A_2, ..., A_k$, where A_i denotes the degree sequence of the part $V_i(G)$. In the sequel, we denote a k-partite degree sequence A as $(A_1 : A_2 : ... : A_k)$ and the sequence $(A_1 : A_2 : ... : A_k)$ is called a k-*partition* of A.

Observation 1 *An easy observation used in the sequel is that, if* $A = (A_1 : A_2 : ... : A_k)$ *is a* k*-partite degree sequence having* n *entries, and* A_i *has* n_i *entries, then the following is true.*

1. $n_1 + n_2 + ... + n_k = n$

2. For every i *such that* $1 \leq i \leq k$*, the maximal entry of* A_i *is less or equal to* $\sum_{j \neq i} n_j$.

The *Degree Sequence Problem* is to find some or all graphs with a given degree sequence [20]. More detailed analysis of the Degree Sequence Problem and its relevance can be found in [18]. Several algorithms are known to construct random realizations of degree sequences and each one of them has its strengths and limitations. Most of these algorithms can be fitted in two categories: MonteCarlo Markov chains methods based on edge-swappings [5, 8, 9, 10, 11, 13, 14, 16, 17] and random matching methods [1, 2, 3, 4, 21]. In particular, algorithms proposed in [1, 3, 7] are based on inserting edges sequentially according to some probability scheme. The basic ideas of the algorithm presented in the present paper have already been used successfully to sample uniformly all the simple realizations of a bipartite degree sequence in [15]. Those basic ideas may be seen as implementing a "dual sequential

method", as it inserts sequentially vertices instead of edges. For other undefined notations and terminology in graph theory, the readers are referred to [19].

Indeed, in the theory of the Tutte polynomial, there are two operations, deletion and contraction, that are dual of each other, see [6] for more details on this topic. Let G be a graph having n vertices and m edges. The operation of deleting the edge $e = (v_i, v_j)$ from G consists of removing the edge e and leaving anything else unchanged. The graph thus obtained, denoted by $G\backslash e$, is a graph on n vertices and $m-1$ edges where h the degrees of both the vertices v_i and v_j go down by 1. The operation of contracting the graph G by $e = (v_i, v_j)$ consists of deleting the edge e and identifying the vertex v_i and v_j. The graph thus obtained, denoted by G/e, is a graph on $n-1$ vertices and $m-1$ edges, where the new vertex obtained by identifying v_i and v_j has degree $a_i + a_j - 2$. Deletion is said to be the dual of contraction as the incidence matrix of $G\backslash e$ is orthogonal to the incidence matrix of G^*/e, where G^* is the dual of G if G is planar.

If A is a degree sequence having n entries, it can be easily shown that random matching methods used in [1, 2, 3, 4, 21] are equivalent to starting from a known realization G of A, delete all the edges one by one, and keeping track of the degrees of vertices after each deletion, until one reaches the empty graph having n vertices. Then, reconstructing a random realization of A consists of taking the reverse of the deletion. That is, starting from the empty graph on n vertices, re-insert edges one by one by choosing which edge to insert according to the degrees of the vertices and some probability scheme depending of the stage where the algorithm is at, and subject to not getting double edges if one would like to get simple graphs or not linking two vertices on the same part if one wants to get bipartite graphs. The algorithm presented in this paper is based on the dual operation of contraction that is slightly modified to suit our purpose. It is equivalent to starting from a known realization G of A, contract all the edges one by one, and keeping track of the vertices after each contraction, until one reaches the graph on one vertex and $\frac{1}{2}\sum_1^n a_i$ loops. Then, reconstructing a random realization of A consists of reversing the process of contraction. That is, starting from the graph on one vertex and $\frac{1}{2}\sum_1^n a_i$ loops, the algorithm re-inserts vertices one by one by choosing which vertex to connect to which according to degrees of the vertices and some probability that depends on the stage of the algorithm.

While algorithms that are based on Markov chains [14, 17] or on reversing the deletion operation [1, 3] are easy to implement, our algorithm seems more complex as one has to satisfy not only the degrees of the vertices, but also

some added graphical structures imposed by the contraction. But this is more of a bonus than an inconvenience, as apart from the fact the the running time is even better, the extra structure allows an easier analysis of the algorithm. Moreover, the internal structure imposed by the contraction operation allows the algorithm to avoid most of the shortcomings of the previous algorithms. Indeed, not only the algorithm never restarts, but the algorithm also allows to sample all bipartite realizations with equal probability, making their approximate counting much easier than by the importance sampling used in [1, 3]. Better still, this technique can be extended, as we do it in the present paper, to construct k-partite realizations of a k-partite degree sequence A, for $k \geq 3$, where a k-partite degree sequence is defined in a natural way by extending the definition of a bipartite degree sequence.

This paper is organized as follows. First we define a recursion chain of a degree sequence, then we present routines for constructing all k-partite realizations. These basic routines are then modified to get a uniform distribution on the set of all k-partite realizations. Then comes the section that presents criteria to generate simple realizations graphs only. We modify our routines to new routines that generate all simple k-partite realizations uniformly at random.

2 Construction of all k-partite realizations of given degrees

2.1 Recursion chain of degree sequences

Let G be a graph with n vertices and m edges. Throughout we assume that the vertices of G are labelled $v_1, v_2, \ldots, v_n$. Let $A = (a_1, \ldots, a_n)$ be the degree sequence of G, where a_i denotes the degree of the vertex v_i. Define an arithmetic operation on A, called *contraction* as follows. For an ordered pair (a_i, a_j) of entries a_i and a_j of A with $i \neq j$, the operation of *contraction* by (a_i, a_j) means changing a_i to $a_i + a_j$ and deleting the entry a_j from A. We write $A/(i, j)$ to denote the new sequence thus obtained. The sequence $A/(i, j)$ is called the (i, j)-*minor* or simply a *minor* of A. The following example illustrates the definition given above for a tripartite degree sequence.

Example 2 *Let* $A = (5, 5 : 4, 2 : 3, 3, 2)$ *where* $a_1 = 5$, $a_2 = 5$, $a_3 = 4$ *and* $a_4 = 2, a_5 = 3, a_6 = 3, a_7 = 2$. *We have* $A/(1, 2) = (10, 4, 2, 3, 3, 2)$ *and* $A/(4, 2) = (5, 4, 7, 3, 3, 2)$.

Let A be sequence of integers. Then A is said to be *graphic* if there is a graph G, not necessarily simple nor k-partite, such that G has A as its degree sequence. Moreover, it is trivial to observe that a sequence of integers is graphic if and only if the sum of its entries is even. Further, we have the following observation.

Theorem 3 *A sequence* A *is graphic if and only if all its minors are graphic.*

Proof. Obviously, if A is graphic, then $A/(a_i, a_j)$ is graphic as, by definition of contraction, the sum of its entries is even. Now suppose that $A/(a_i, a_j)$ is graphic and G'' is a realization of $A/(a_i, a_j)$. To prove that A is also graphic, we present an algorithm, much used in the sequel, that constructs a realization of A, denoted by G, from G''.

Algorithm AddVertex()
Step 1. To G'' add an isolated vertex labelled v_j (as in Figure 1).
Step 2 If the degree of v_j is a_j, stop, output G. Else
Step 3. Amongst the a'_i edges incident to v_i, counting loops twice, choose one edge
$e = (v_i, v_k)$ with probability $\pi(e)$ and connect e to v_j so that e becomes (v_j, v_k). Go to Step 2.

Now, in G the degree of v_j is a_j, by Step 2 of algorithm AddVertex(). Moreover, by the definition of contraction, the degree of v_i is equal to $a_i + a_j$ in G''. Since AddVertex() takes a_j edges away from v_i, the degree of v_i is a_i in G. Moreover all the other vertices are left unchanged by AddVertex(). Thus G is a realization of A. □

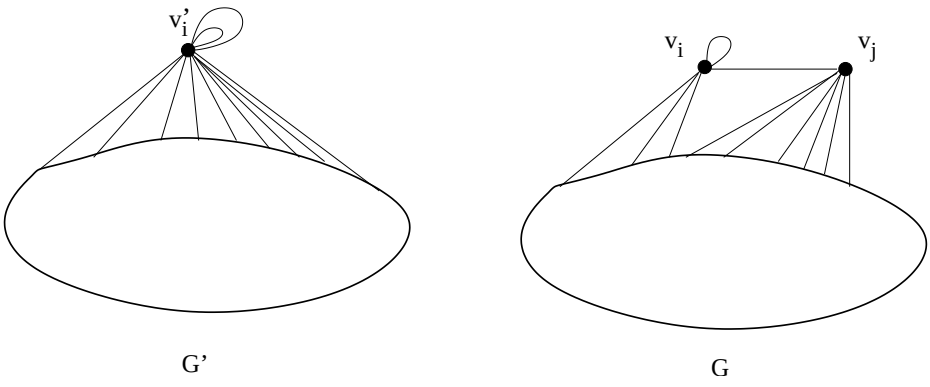

Figure 1: Construction of a graph G from its minor G''

If AddVertex() chooses the edge $e = (v_i, v_k)$ and connects e to v_j so that e becomes (v_j, v_k), we say that AddVertex() (or v_i, or v_k) concedes e to v_j. To help intuition, observe that if G'' is a realization of $A/(a_i, a_j)$ and G is a realization of A constructed by AddVertex(), then G'' is obtained from G by contraction of the edge (v_i, v_j). Now, mimicking the process of recursive contraction of matroid as used in the theory of the Tutte polynomial, we define a process of recursive contraction for a degree sequence. A *recursion chain* of a degree sequence A is a unary tree rooted at A, where nodes are integer sequences and every node, except for the root, is a minor of the preceding one. The recursive procedure of contraction is carried on from the root A until a node with a single entry is reached. As for the Tutte polynomial, the amazing fact, which is then used to construct all the realizations of A, is that the order of contraction is immaterial. Despite this basic fact, we still impose a particular order to ease many proofs in the sequel.

Notes on notations: For the sake of convenience, we refer to a node of a recursion chain of a degree sequence A by $A^{(i)}$, where i is the number of entries in the node. Thus we denote the root A by $A^{(n)}$, the next node by $A^{(n-1)}$, and so on until the last node $A^{(1)}$. Similarly, we denote by $G^{(i)}$ the realization of $A^{(i)}$. The n entries of A are labelled from 1 to n. To keep tract of the vertices, we preserve the labelling of entries of A into its minors so that when a contraction by the pair (a_i, a_j) is performed, the new vertex is labelled a_i, the label a_j is deleted, and all the other entries keep the labelling they have before the contraction.

In this paper, we consider the recursion chain, called the k-*partiteaccumulating recursion chain*, constructed as follows. Let $A = (A_1 : A_2 : ... : A_k)$ be a k-partite degree sequence. We label each entry as $a_{s,r}$, where s ranges from 1 to n and r ranges from 1 to k, so that the entry $a_{s,r}$ belongs to A_r.

Example 4 *Let* $A = (5,5 : 4,2 : 3,3,2)$. *We order entries of* A *as* $a_{1,1} = 5$, $a_{2,1} = 5$, $a_{3,2} = 4$, $a_{4,3} = 3$, $a_{5,3} = 3$, $a_{6,2} = 2$, $a_{7,3} = 2$. *Thus* $\vec{A} = (5,5,4,3,3,2,2)$.

Note. The vertex having degree $a_{s,r}$ is denoted by $v_{s,r}$. But, to avoid clustering the notation, we sometimes just right a_s or v_s, when we deem not necessary to specify the part A_r corresponding to the degree $a_{s,r}$.

Algorithm ConstructKpartiteRecursionChain()
Given an ordered k-partite degree $\vec{A}$. Let $i = n$.
Step 1 If $i = 1$, stop, return $\{A^{(1)}, A^{(2)}, ..., A^{(n)}\}$. Else
Step 2 Let $A^{(i-1)} = A^{(i)}/(1, i)$. That is, get the $(i-1)^{th}$ recursive minor of A by contracting the $(i)^{th}$ recursive minor by its first entry and the last entry.
Step 3 Decrement i by 1 and go back to Step 1.

We denote the accumulation recursion chain of $\vec{A}$ by $W = (A^{(1)}, A^{(2)}, ..., A^{(n)})$.

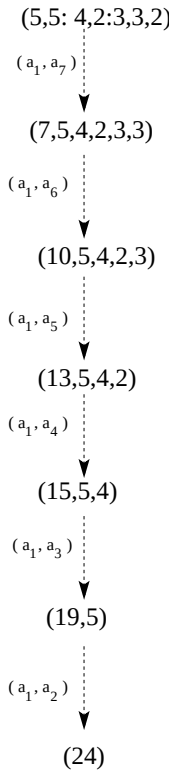

Figure 2: The accumulating recursion chain of the tripartition $[5, 5 : 4, 2 : 3, 3, 2]$.

The following is an algorithm for constructing a k-partite realization if A is a k-partite degree sequence. The graph constructed below is not necessarily simple. Loosely speaking, this algorithm consists of reversing the recursive process of contraction as implemented by ConstructKpartiteRecursionChain(). The algorithm starts from $G^{(1)}$ the sole realization of $A^{(1)}$, and by calling

AddVertex() recursively, it constructs $G^{(2)}$, then $G^{(3)}$, and so on until $G^{(n)}$, that is a realization of $A^{(n)} = A$. The only condition imposed on the choice to built edges is that if the vertex to insert has degree $a_{s,r}$, then we label the vertex as $v_{s,r}$. Then AddVertex links a vertex $v_{x,y}$ to $v_{s,r}$ only if $j \neq r$, unless $x = 1$. Recall that $\delta(x, y) = 1$ if $x = y$ and $\delta(x, y) = 0$ otherwise.

Algorithm ConstructKpartiteRealization()

Given $W = (A^{(1)}, A^{(2)}, ..., A^{(n)})$, the Kpartite accumulating recursion chain of A, we do the following.

Step 1. Let $i = 1$ and build the realization of the node $A^{(1)}$, denoted by $G^{(1)}$, which is the graph consisting of one vertex and m loops, where $m = \frac{1}{2}\sum_{i=1}^{n} a_i$.

Step 2. Let $G = G^{(i)}$. If G has n vertices, stop, return G. Else,

Step 3. Using $G^{(i)}$ and $A^{(i+1)}$ as input, Call Algorithm AddVertex() to construct $G^{(i+1)}$ as a realization of $A^{(i+1)}$. If $v_{s,r}$ is the vertex being inserted, then AddVertex only constructs edges $(v_{x,y}, v_{s,r})$ where $\delta(y, r) = 0$ and $\delta(x, s) = 0$, unless $x = 1$. Increment i by 1, go back to Step 2.

See Figure 3 for an illustration of Algorithm ConstructBipartiteRealization().

The following definitions are needed in the sequel. In the process of contraction implemented by the accumulating recursion chain, we observe that the degrees are accumulating on $a_{1,1}$. This is equivalent to say that edges are accumulating on $v_{1,1}$ as $v_{1,1}$ seems to 'swallow' the other vertices one by one. Hence, when reversing the contraction operation in ConstructKpartiteRealization(), vertex $v_{1,1}$ plays the role of the 'mother that spawns' all the other vertices one by one and concedes some edges to them according to their degrees. Thus, AddVertex() can attach an edge e to a new vertex $v_{s,r}$ only if e is incident to $v_{1,1}$. This observation prompts the following formal definitions. Let A be a k-partite degree sequence where A_i has n_i entries such that $\sum_i n_i = n$. The $(s, t)^{th}$ stage of ConstructKpartiteRealization() is the iteration where the algorithm inserts the t^{th} edge of the vertex $v_{s,r}$. At the $(s, t)^{th}$ stage an edge is *available* if it is a loop incident to $v_{1,1}$ or $e = (v_{1,1}, v_{x,y})$ where $y \neq r$ and $x < s$. An edge e is *lost* otherwise. Let E_{av} denote the set of all available edges and E_{v_j} the set of edges (v_1, v_j). We recall that an edge $e = (v_{1,1}, v_{x,y})$ is *conceded* if AddVertex() disconnects it from $v_{1,1}$ so that e becomes $e = (v_{x,y}, v_{s,r})$ for some vertex $v_{s,r} \neq v_{1,1}$. We then say that $v_{1,1}$ (or sometimes E_{v_j} or just v_j) concedes the edge e. A vertex v_s having degree a_s is *fully inserted* if a_s edges are conceded to it. A graph G is said to be *(re)constructed* if it is an output of

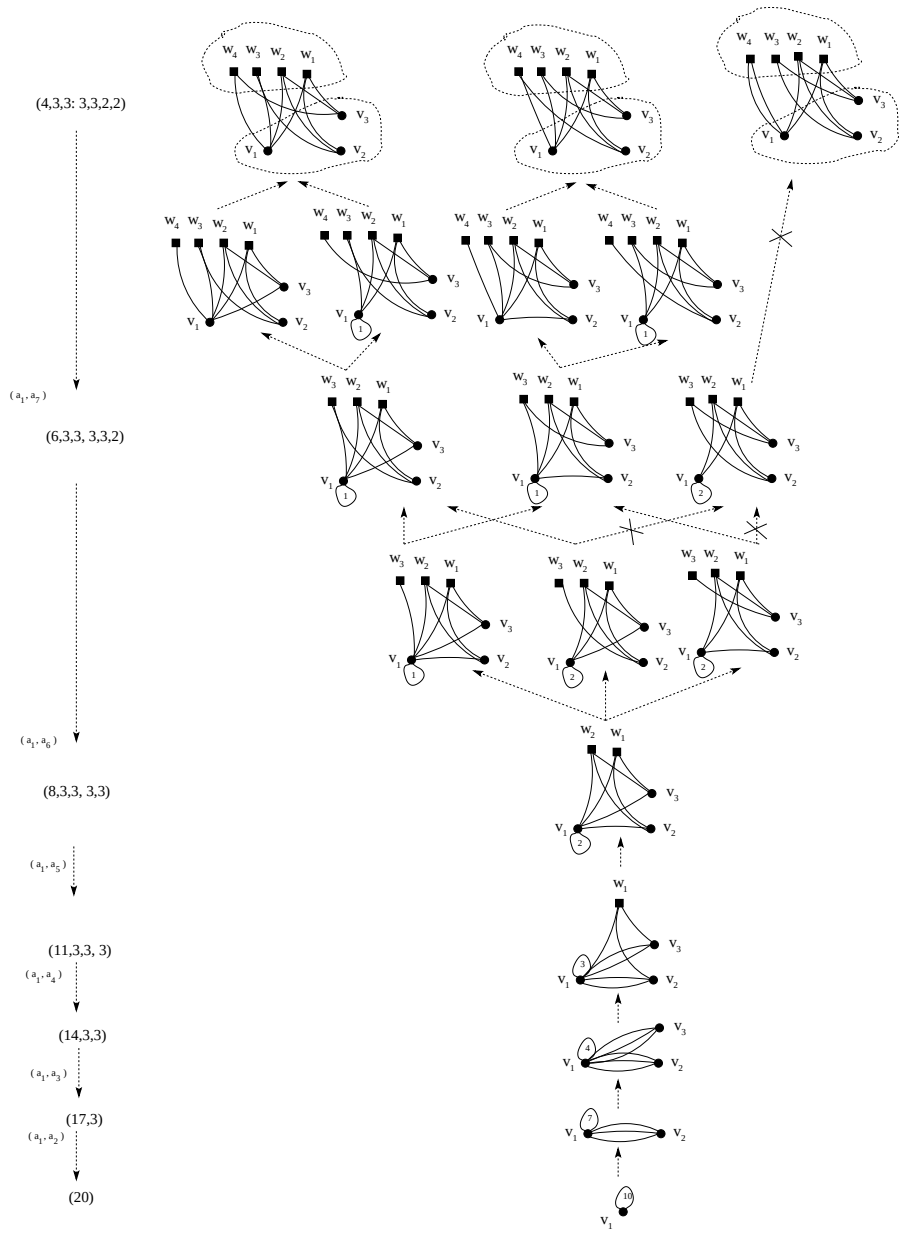

Figure 3: Random reconstruction tree of $(4,3,3:3,3,2,2)$. The level of $\mathcal{T}$ on the same height as the degree sequence $A^{(i)}$ corresponds to all the graphs having $A^{(i)}$ as their degree sequence.

ConstructKpartiteRealization(). Now we state and prove a paramount result of this paper.

Theorem 5 *Let* $A = (a_1, a_2, ..., a_n) = (A_1 : A_2 : ... : A_k)$ *be a* k*-partite degree sequence having* n *entries where* A_i *has* n_i *entries, such that* $\sum n_i = n$ *and* $m = \frac{a_i + a_2 + ... + a_n}{2}$ *. Let* W *be the* k*-partite recursion chain of* A*. Then Algorithm ConstructKpartiteRealization() constructs in time linear on* m *a* k*-partite graph* G *having* n *vertices and* m *edges such that* G *is a realization of* A*. Moreover, every* k*-partite realization of* A *can be constructed in this way.*

Proof. By Algorithm AddVertex(), the graph $G^{(n)}$ output by Algorithm ConstructKpartiteRealization() is assured to be a realization of A. We need only to prove that $G^{(n)}$ is k-partite. Now, the routine AddVertex() constructs edges $(v_{s,x)}, v_{r,y}$ only if $\delta(s, r) = \delta(x, y) = 0$, unless $s = x = 1$. Thus vertices corresponding to degrees in different parts A_x and A_y are never adjacent. Thus, we only have to show that in G^n, $v_{1,1}$ is not adjacent to any vertex $v_{j,1}$. We need the following fact.

Observation 6 *Suppose that* A *is a* k*-partite degree sequence. From the* v_1^{th} *iteration of ConstructKpartiteRealization(), the number of available edges is equal to the number of edges left to be inserted until ConstructKpartiteRealization() terminates.*

This is because the number of available edges at the end of $(v_1)^{th}$ is equal to half the sum of degrees $a_i \in A_1$. By the definition of the bipartite degree sequence, this number is equal to half the sum of degrees $a_j \in A_2$

Now, to prove that $G = G^n$ is k-partite, in the proof of Theorem 5, suppose for a contradiction, G contains an edge $e = (v_{1,1}, v_{j,1})$. Consider the graph H obtained from G by contracting all the edges not incident to any vertex $v_{j,1}$. It is easy to see that the degree sequence of H, denoted by B, is the degree sequence obtained from A by contracting the entries corresponding to vertices contracted in G. Thus B is a k-partite degree sequence. Now, suppose that ConstructKpartiteRealization() outputs a realization K of B with en edge $(v_{1,1}, v_{j,1})$. Then by Observation 6 one vertex of K is not fully inserted which is a contradiction. Thus ConstructKpartiteRealization() reconstructs H has no edge $(v_{1,1}, v_{j,1})$. This is the final contradiction we are looking for.

It remains to prove that any k-partite realization of A can be constructed this way. We recall that v_i (a_i) denotes the i^{th} vertex (degree) in the ordering, regardless of the second index. So, let G be a realization of A and let $e = (v_i, v_j)$ be any edge of G such that vertex v_i has degree a_i and vertex v_j has degree

a_j. Also, suppose that the vertcies v_i and v_j were inserted at the i^{th} and j^{th} iteration of ConstructKpartiteRealization() respectively. We need to show that at the j^{th} iteration, there is a positive probability to have an edge e which is incident to v_i and e is available. If not, that is, at the j^{th} iteration all the edges incident to v_i are lost. Now all the edges incident to v_i are lost before the j^{th} iteration only if, at some stage of the running of Algorithm ConstructKpartiteRealization(), there are only the edges that are available and they are exhausted before reaching the j^{th} iteration. Thus, at the j^{th} iteration there are no more available edges. Especially, there are no loops incident to v_1. But this means that $a_1 + a_2 + ... + a_j \geq 2m$, contradicting the definition of accumulating recursion chain.

As for the running time, Algorithm ConstructKpartiteRealization() calls Algorithm AddVertex() once for every new vertex v_k to insert. If v_k has degree a_k, then Algorithm AddVertex() has to go through a_k iterations to insert the a_k edges of v_k. Hence the total number of iterations to terminate ConstructKpartiteRealization() is $a_1 + a_2 + ... + a_n = 2m$. □

2.2 Sampling all k-partite realizations uniformly

Although Theorem 5 shows that the routine ConstructKpartiteRealization() can construct a realization of A in linear time, we need the next result to show that it can construct any k-partite realization of A with equal probability, provided we define the probability $\pi(e)$ with which AddVertex() has to insert the edge e.

We recall that if at its s^{th} iteration ConstructKpartiteRealization() is to insert the vertex v_s that has degree a_s, (regardless of the second index of v_s), then ConstructKpartiteRealization() has to *call* AddVertex() that has to go through a_s iterations. Let the $(s,t)^{th}$ stage of ConstructKpartiteRealization() be the iteration, where AddVertex() inserts the t^{th} edge of the s^{th} vertex and let $G^{(s,t)}$ denote the graph obtained at that $(s,t)^{th}$ stage. With this notation, let $G^{(s)}$ be the graph $G^{(s,a_s)}$.

The *random reconstruction tree*, denoted by $\mathcal{T}$, is a directed rooted tree, where the root is the sole realization of the degree sequence $A^{(1)}$, and the $(s,t)^{th}$ level contains all those possible graphs obtainable after inserting the t^{th} edge of the s^{th} vertex, and there is an arc from a graph H at level i to the graph G at level $i+1$ if it is possible to move from H to G by the concession of a single available edge. Realizations of A are thus the leaves of the tree $\mathcal{T}$. With this formalism, sampling a random k-partite realization of the degree sequence A is equivalent to performing a random walk from the root until a

leaf is reached, and every step of the random walk consists of walking along a random arc of $\mathcal{T}$. See Figure 3 for an illustration.

Observations about uniform sampling

(1) Let G^1 denote the root of $\mathcal{T}$ and G^1 be the graph on a single vertex and m loops. So the stage $(2,1)$ of ConstructKpartiteRealization consists of inserting the first edge of second vertex v_2. Suppose at some stage (s,t) of ConstructtKpartiteRealization(), the random walk along $\mathcal{T}$ is at the graph $G^{(s,t)}$ with probability $\pi(G^{(s,t)})$ and that AddVertex() is to concede an edge e to v_s. Suppose also that the vertex v_j of $G^{(s,t)}$ has $|E_{v_j}|$ available edges. That is, v_j is connected to v_1 by $|E_{v_j}|$ parallel edges. Thus, the next level of $\mathcal{T}$ would contain $|E_{v_j}|$ identical graphs whose edge sets will contain the edge (v_j, v_s). Hence, if Algorithm AddVertex() chooses every available edge uniformly at random, the random walk will reach such a graph with probability $\frac{|E_{v_j}|}{|E_{av}|}$. Thus, if this graph is a leaf of $\mathcal{T}$, ConstructBipartiteRealization() will be biased toward it with a probability proportional to $\frac{|E_{v_j}|}{|E_{av}|}$. So, if the random walk has to reach each different child of $G^{(s,t)}$ with the same probability, we have to move from $G^{(s,t)}$ to its child obtained by adding the edge $e \in E_{v_j}$ with probability $\frac{|E_{v_j}|}{|E_{av}|}\frac{1}{|E_{v_j}|} = \frac{1}{|E_{av}|}$. Equivalently, let $|V(G^{(s)})|$ be the number of vertices already inserted up to the s^{th} iteration of ConstructKpartiteRealization() and let $|V'(G^{(s)})|$ be the number of vertices in $V(G^{(s)})$ which are adjacent to v_1. Obviously $v_1 \in V'(G^{(s)})$, if there is a loop incident to v_1. We may choose any vertex $v_j \in V'(G^{(s)})$ uniformly at random. If $e \in E_{v_j}$, then we concede e with probability $\frac{|V'(G^{(s)})|}{|V(G^{(s)})|}$.

(2) Suppose that $G^{(s,t+1)}$ is obtained from $G^{(s,t)}$ by the concession of edge e to vertex v_s. If in $G^{(s,t)}$ the vertex v_s is adjacent to b_s (with $b_s \leq a_s$) different vertices, then it is obvious that the random walk on $\mathcal{T}$ reaches $G^{(s,t)}$ in b_s different paths. Thus there is a bias proportional to b_s towards $G^{(s,t)}$. To remove this bias, the random walk would rather move away from $G^{(s,t)}$ with a reducing factor of $\frac{1}{b_s}$. Similarly $G^{(s+1,1)}$ is obtained from $G^{(s,a_s)}$ by the concession of edge e to vertex v_{s+1}. These two observations prompt the following extension of the routines Addvertex() and ConstructKpartiteRealization() to sample uniformly.

Algorithm BiasedAddvertex()

(A modification of Algorithm Addvertex() to get a uniform distribution, dividing by the number of vertices inserted up to the s^{th} iteration).
Step 1 To the graph G^s, add an isolated vertex called v_{s+1}. Let a_{s+1} be the number of edges to concede to v_{s+1} and let $j = 0$.
Step 2 If v_{s+1} is incident to a_{s+1} edges, return G^{s+1}. Else,
Step 3 Let b^i_{s+1} be the number of different vertices v_i which are incident to v_{s+1} after the insertion of its j^{th} edge, with $j \geq 1$. If $j = 0$, let b^j_{s+1} be the number of different vertices adjacent to v_s.
Step 4 Choose vertex v_q uniformly at random amongst all the vertices adjacent to v_1.
Step 5 If $e \in E_{v_q}$, then concede e to v_{s+1} with probability $\frac{|V'(G^{(s)})|}{|V(G^{(s)})|.b^j_{s+1}}$, where $V(G^{(s)})$ and $V'(G^{(s)})$ are respectively the set of vertices inserted up to the s^{th} iteration and the set of vertices in $V(G^{(s)})$ that are adjacent to v_1. Increment j by 1 and go back to Step 2.

Accordingly, we modify the routine ConstructBipartiteRealization() as follows.

Algorithm BiasedConstructKpartiteRealization()

Given the k-partite recursion chain of $A = (A_1 : A_2 : ... : A_k)$, where A_i has n_i entries such that $n_1 + n_2 + ... + n_k = n$, do the following.
Step 1. Let $s = 1$ and build the realization of the node $A^{(1)}$, denoted by $G^{(1)}$, that is the graph consisting of one vertex and m loops.
Step 2. Let $G = G^{(s)}$. If G has n vertices, stop, return G. Else,
Step 3. Using $G^{(s)}$ and $A^{(s+1)}$ as input, call Algorithm BiasedAddVertex() to construct $G^{(s+1)}$ as a realization of $A^{(s+1)}$. If $v_s = v_{s,y}$ BiasedAddvertex only concedes loops or edges $(v_{1,1}, v_{r,x})$ such that $\delta(x, y) = 0$. Increment s by 1, go back to Step 2.

Theorem 7 *(1) For the degree sequence* $A = (A_1 : A_2 : ... : A_k)$*, where* A_i *has respectively* n_i *vertices such that* $n_1 + n_2 + n_k = n$*, BiasedConstructKpartiteRealization() reaches every leaf of* $\mathcal{T}$ *uniformly at random with probability* $\frac{1}{\prod_{s=1}^{n} s^{a_{s+1}}}$.
(2) The set of leaves of $\mathcal{T}$ *is the set of realizations of* A

Proof. (1) Suppose that BiasedConstructKpartiteRealization() calls BiasedAd-

dVertex() to insert vertex v_s that has degree a_s, where $s > n_i$, and that BiasedAddvertex() is conceding the j^{th} edge of v_s. We know there are $|V(G^{(s)})|$ vertices inserted up to the s^{th} iteration. Now, if $e \in E_{v_t}$, then the routine BiasedAddVertex() will choose e with probability $\frac{1}{|V'(G^{(s)})|}$ and would concede it with probability $\frac{|V'(G^{(s)})|}{|V(G^{(s)})|.b_s^{j-1}}$. But the graph $G^{(s,j-1)}$ is reached through b_s^{j-1} different paths. Hence the graph $G^{(s,j)}$ obtained by conceding e to v_s will be sampled with probability given as

$$\frac{b_s^{j-1}}{|V'(G^{(s)})|}.\frac{|V'(G^{(s)})|}{|V(G^{(s)})|.b_s^{j-1}} = \frac{1}{|V(G^{(s)})|} = \frac{1}{s-1}.$$

Finally, we know that vertex v_s needs a_s iterations of BiasedAddVertex to be fully inserted. Hence, it takes altogether $\sum_{i=1}^{n} a_i = 2m$ iterations before BiasedConstructBipartiteRealization() terminates. Thus multiplying all the probabilities to move from one level to the next from the root to a leaf yields probability $\frac{1}{1^{a_1}2^{a_2}...n^{a_n}}$.

(2) Obviously, by construction, every leaf of $\mathcal{T}$ is a realization of A. The proof that every realization of A is a leaf of $\mathcal{T}$ is given in the proof of Theorem 5. □

3 Construction of simple k-partite graphs

Up to now, BiasedConstructKpartiteRealization() generates all the k-partite realizations of the k-partite degree sequence A. But, it is easy to modify BiasedAddVertex() so that the output of BiasedConstructKpartiteRealization() is always a simple graph. One obvious condition is stated as follows.

(a) If the Algorithm is inserting the j^{th} edge of vertex v_s with $j > 1$ and $s > n_i$ with $1 \leq i \leq k$ and v_l is already adjacent to v_s, then no more available edge incident to v_l should be chosen. This will prevent BiasedConstructKpartiteRealization() from outputting graphs with multiple edges (v_s, v_l). Thus this condition is necessary but is not sufficient. Indeed, it is easy to see that the following must also apply.

(b) While inserting vertex v_s and avoiding choosing edges incident to v_l so as not to construct multiple edges (v_s, v_l), BiasedConstructKpartiteRealization() may fall into a stage where there are more edges incident to v_l than there are vertices left to be inserted, and G, the graph output by BiasedConstructBipartiteRealization() will then have a multiple edge (v_l, v_l).

Although (a) and (b) seem to contradict each other, this section defines all these conditions in a formal settings and proves that they can be satisfied simultaneously. Although the analysis seems long, this set of conditions are just inequalities involving the number of edges and vertices already inserted and the number of edges and vertices left to insert at each stage of the Algorithm. Moreover, checking these conditions at each iteration of BiasedAddVertex() requires checking $\mathcal{O}(n^2)$ inequalities altogether. Thus it does not add crucially to the running time.

Let $A = (A_1 : A_2 : ... : A_k)$ be a k-partite degree sequence of a simple graph, where A_i has n_i vertices such that $\sum n_i = n$. Suppose that BiasedConstructKpartiteRealization() is at the iteration of inserting vertex v_s. We recall that E_{av} represents the set of available edges at the $(s,t)^{th}$ stage. That is, edges that are incident to v_1 and vertices inserted before the iteration inserting the t^{th} edge of the s^{th} vertex. We also recall that E_{v_j} are the edges (v_1, v_j) at that stage. In particular, E_{v_1} is the set of loops incident to v_1. The set of *excess edges* of v_j, denoted by Ee_{v_j}, is the same as the set E_{v_j} if $v_j = v_{j,1}$. In particular, a loop is an excess edge incident on v_1. If $v_j = v_{j,r}$ for $r \neq 1$, the set of *excess edges* of v_j is the set E_{v_j} except one edge. That is $|Ee_{v_j}| = |E_{v_j}| - 1$.

The aim of this section is to show that it is possible to choose edges so that the algorithm never stalls after choosing a 'wrong' edge. If at its s^{th} iteration, Algorithm BiasedConstructKpartiteRealization() is inserting the vertex v_s that has degree a_s, then BiasedConstructKpartiteRealization() has to *call* the routine BiasedAddVertex() which has to go through a_s iterations. We recall that the $(s,t)^{th}$ stage of BiasedConstructKpartiteRealization() is the iteration, where BiasedAddVertex() inserts the t^{th} edge of the s^{th} vertex. Let $X_{s,t}$ and $|X|_{s,t}$ respectively denote a set and its cardinality at the $(s,t)^{th}$ stage of BiasedConstructKpartiteRealization().

To help the reader, we first introduce the motivation behind every definition. Obviously, if at some stage, the number of excess edges is greater than the number of edges left to be inserted, then BiasedConstructKpartiteRealization() can never produce a simple graph as the left-over of excess edges would result in a multiple edge or a loop in the final graph. Thus the choice of edges by BiasedAddvertex() must be such that this contingency never happens. This prompts the following definitions.

The $(s,t)^{th}$ stage of Algorithm BiasedConstructKpartiteRealization() is *critical* if

$$|Ee|_{st} = a_s - (t-1) + a_{s+1} + a_{s+2} + ... + a_n. \tag{1}$$

That is, the number of excess edges equals the number of edges left to insert until the end of BiasedConstructRealization(). The $(s,t)^{th}$ stage is *spoilt* if

$$|Ee|_{st} > a_s - (t-1) + a_{s+1} + a_{s+2} + ... + a_n. \tag{2}$$

That is, there are too many excess edges and whatever the future choices might be, a simple graph can never be output. A stage is *normal* if it is neither critical nor spoilt.

Now, at each stage of constructing a simple k-partite graph, every vertex $v_{r,x}$ must be connected to any other $v_{s,y}$, with $\delta(x,y) = 0$ unless $r = x = 1$, by at most one common edge. So, if some vertex $v_{r,x}$, with $r \neq 1$, has more excess edges than the number of vertices $v_{s,y}$, with $\delta(x,y) = 0$, left to be inserted, BiasedConstructKpartiteRealization() would never be able to get rid of all these multiple edges, which will then appear in the final graph. This prompts the following definition. Let $N(\bar{x})_{st}$ be the the set of vertices $v_{q,y}$ with $q < s$ and $y \neq x$. At the $(s,t)^{th}$ stage, the vertex $v_{r,x}$ with $r \leq s$ and $r \neq 1$ is *due* if

$$|Ee_{v_{r,x}}|_{st} = |N(\bar{x})|_{st}, \tag{3}$$

that is, $Ee_{v_{r,x}}$ has got as many excess edges as there are vertices left to be inserted to which it can concede an edge. The vertex $v_{r,x}$ is *overdue* if

$$|Ee_{v_{r,x}}|_{st} > |N(\bar{x})|_{st}, \tag{4}$$

that is, there are too many excess edges incident to $v_{r,x}$ and whatever the future choices might be, the Algorithm will never output a simple graph. The vertex $v_{r,x}$ is *undue* if it is neither due or overdue.

Now, although v_1 may concede many edges to a vertex v_s, there is also a limit, other than a_s, to the number of edges it can concede to v_s if the end result is to be a simple graph. For example, BiasedAddvertex() can construct at most 1 edge (v_1, v_n), by conceding a loop incident to v_1 to the vertex v_n. Similarly BiasedAddvertex() can construct at most 2 edges (v_1, v_{n-1}) . Otherwise, these vertices will be overdue. More generally, BiasedAddvertex() can construct at most q edges (v_1, v_{n-q+1}). This requirement prompts the following definitions.

The vertex v_1 is *due* if

$$|Ee_{v_1}|_{st} = 1 + 2 + ... + (n-s) + (a_s - t),$$

that is, Ee_{v_1} has got just enough loops to make each of the remaining vertices due.

The vertex v_1 is *overdue* if

$$|Ee_{v_1}|_{st} > 1 + 2 + ... + (n - s) + (a_s - t).$$

Similarly to other vertices, v_1 is *undue* if it is neither due nor overdue.

Moreover, if the degree sequence A has no entry $a_i = 1$, then the vertex v_1 is *ripe* if

$$|Ee_{v_1}|_{s,t} = \sum_{j \neq 1} |Ee_{v_j}|_{s,t}. \tag{5}$$

It is *overipe* if

$$|Ee_{v_1}|_{s,t} > \sum_{j \neq 1} |Ee_{v_j}|_{s,t}. \tag{6}$$

If the degree sequence A has an entry $a_i = 1$, then the vertex v_1 is *ripe* if

$$|Ee_{v_1}|_{s,t} = \sum_{j \neq 1} |Ee_{v_j}|_{s,t} - 1. \tag{7}$$

It is *overipe* if

$$|Ee_{v_1}|_{s,t} > \sum_{j \neq 1} |Ee_{v_j}|_{s,t} - 1. \tag{8}$$

In both cases, this means that there are more loops than all the other excess edges put together, and so, if the stage is also critical, whatever the future choices might be, the Algorithm will never produce a simple graph, as there will be at least one loop left incident to v_1. The vertex v_1 is *unripe* if it is neither ripe nor overipe. We also say that a stage is *due* (*overdue*, *undue*, *ripe*, *overipe*, *unripe*) if it contains a vertex that is due (overdue, undue, ripe, overipe, unripe). It should be understood that saying that E_{v_i} is *due* (*overdue*, *undue*, *ripe*, *overipe*, *unripe*) only means that v_i is *due* (*overdue*, *undue*, *ripe*, *overipe*, *unripe*).

The next lemma only means that once BiasedConstructKpartiteRealization() has taken a 'wrong path', it is impossible to mend the situation.

Lemma 8 *Suppose that BiasedConstructKpartiteRealization() is inserting the vertex $v_{s,y}$ and suppose that BiasedAddvertex() satisfies the following condition.*

Condition (1) For each vertex $v_{r,x}$ with $\delta(x, y) = 0$ and $r \neq 1$, BiasedAddvertex() must choose at most one edge from E_{v_r} so that there is never a double edge (v_r, v_s).

Then the following hold.

(a) If the $(s,t)^{th}$ *stage is critical, then the next stage is critical or some future stage is spoilt.*

(b) If the $(s,t)^{th}$ *stage is spoilt, then any future stage is spoilt.*

(c) If the vertex $v_{r,x}$ *is due, it is due or overdue at the next stage. If it is overdue, it is overdue at any future stage.*

(d) If the $(s,t)^{th}$ *is critical and* v_1 *is overripe, then the last stage is spoilt.*

(e) If the $(s,t)^{th}$ *stage is spoilt, then the previous stage (the stage inserting the previous edge) is either spoilt or critical.*

(f) If the $(s,t)^{th}$ *stage is overipe, then the previous stage (the stage inserting the previous edge) is either overipe or ripe.*

(g) If the $(s,t)^{th}$ *stage is overdue, then the previous stage (the stage inserting the previous edge) is either due or overdue.*

Proof. At the $(s,t)^{th}$ stage, where the t^{th} edge of the vertex $v_{s,y}$ the following types of edges are available.

(1) A loop incident to v_1 and, if conceded, the resulting edge (v_1, v_s) is single,

(2) a loop incident to v_1 and, if conceded, the resulting edge (v_1, v_s) is a multiple edge,

(3) a single edge $(v_1, v_{r,x})$ for some $r < s$ and $\delta(x,y) = 0$ or

(4) a multiple edge $(v_1, v_{r,x})$ for $r < s$ and $\delta(x,y) = 0$.

(a) In case of choice (1), both the right side and the left side of Equation 1 go down by one. Thus the next stage is still critical. In case of choice (2), the resulting edge (v_1, v_s) forms a multiple edge with a previously inserted edge. Thus, the next stage is spoilt as the left hand side stays the same while the right hand side goes down by 1. If Algorithm BiasedAddvertex() chose an edge of type (3) then the left side of Equation 1 will stay the same while the right side goes down by 1, hence the next stage would be spoilt. If BiasedAddvertex() chose an edge of type (4), then both the right side and the left side of Equation 1 will go down by one. Thus the next stage will still be critical. Thus, whatever the choice, the next stage is either critical or spoilt.

(b) Using the same arithmetic arguments as above, it is easy to see that if a stage is spoilt, the next stage is spoilt.

(c) Suppose that $j \neq 1$, the vertex v_j is due and $v_j = v_{j,x}$. If $v_s \notin N(\bar{x})_{st}$, then the next step is due. So, suppose that $v_s \in N(\bar{x})_{st}$. If BiasedAddvertex() makes the choice (3) or (4) from E_{v_r} with $r \neq j$, then since no edge of E_{v_j} is chosen, the left side of Equation 3 stays the same while the right hand side either goes down by one if BiasedConstructKpartiteRealization() moves to a new vertex

v_{s+1} or stays the same if the algorithm moves to another edge $t+1$ of the same vertex v_s. Hence the next stage is due or overdue. If BiasedAddvertex() makes the choice (4) by choosing an edge from E_{v_j}, then both sides go down by 1 and the next stage is due. Obviously, if BiasedAddvertex() makes the choice (1) or (2), v_j stays due or becomes overdue since in any case the left side of Equation 3 stays the same while the right side either goes down by one if the algorithm moves to a new vertex v_{s+1} or stays the same if the algorithm moves to a new edge of the same vertex v_s.

If v_j is overdue, it stays overdue since, for any choice, Condition (1) makes the right side of Equation 4 to go down by 1 while the left side may go down by 1 or stays the same.

A similar argument, replacing loop by edge of type (3) or (4), and vice versa, holds for the case where v_1 is due.

Suppose the vertex v_s to be inserted, is due. If BiasedAddvertex() chose a loop, the left hand goes up by 1 while the right hand side of Equation 3 stays the same. Thus v_s is overdue at the next stage. If an edge of type (3) or (4) is chosen, then both the left and the right hand sides of Equation 3 stay the same. Thus v_s is due at the next stage.

(d) By (a), the future stages will be either critical or spoilt. Suppose that there are more loops than other excess edges. If two loops are conceded to the same vertex, then the next stage is spoilt. So suppose v_1 concedes at most one loop to each of the remaining vertices. BiasedAddvertex() is then forced to pick edges from other vertices. If it picks an edge of type (3), the next stage is spoilt. So it must pick edges of type (4) only. But then edges of type (4) will be exhausted before the loops. Hence there will be at least one vertex that must conceded two loops. Thus the last stage will be spoilt.

(e) Suppose that the $(s,t)^{th}$ stage is spoilt but the previous stage (the stage inserting the previous edge) is normal. Then at the previous stage we have

$$|Ee|_{st} < a_s - (t-1) + a_{s+1} + a_{s+2} + \ldots + a_n. \tag{9}$$

The insertion of one edge always lowers the right hand side of Equation 9 by 1 while the left hand side is the same if BiasedAddvertex() chooses an edge of type (2) or (3) or is lowered by 1 if an edge of type (1) or (4) is chosen. Hence the $(s,t)^{th}$ stage is either critical or normal. This is a contradiction.

(f) Suppose that A has no entry $a_i = 1$ and suppose that v_1 is overipe at the $(s,t)^{th}$ stage but is unripe at the stage inserting the previous edge. That is, at that previous stage we have

$$|E_{v_1}| < \sum_j |Ee_{v_j}|. \tag{10}$$

Now, the last edge inserted is of type (1), (2), (3) or (4). If the chosen edge is of type (1) or (2) or (3), then Equation 10 is unchanged. Hence v_1 is unripe at the $(s,t)^{th}$ stage. This is a contradiction. If the chosen edge is of type (4), then the right hand side of Equation 10 goes down by 1 while the left hand side is unchanged. Hence v_1 is ripe at the $(s,t)^{th}$ stage and this is also a contradiction. The argument is similar if A has an entry $a_i = 1$

(g) Suppose $v_{j,x}$ is overdue at the $(s,t)^{th}$ stage but is undue at the stage inserting the previous edge. Then at the previous stage, we have

$$|Ee_{v_{j,x}}| < |N_{\bar{x}}|. \tag{11}$$

Now, the last edge inserted is of type (1) or (2) or (3) or (4). Moreover, in either case, BiasedConstructKpartiteRealization() moves to a new vertex or not. If it stays on the same vertex and the chosen edge of type (1) or (2) or (3), then the right and the left hand sides of Equation 11 are both unchanged. Hence v_j is undue at the $(s,t)^{th}$ stage. This is a contradiction. If the chosen edge is of type (4) from Ee_{v_j}, then the left hand side of Equation 11 goes down by 1 while the right hand side is unchanged. Hence v_j is also undue at the $(s,t)^{th}$ stage and this is also a contradiction. If the chosen edge is of type (4) from Ee_{v_i} with $i \neq j$, then both the left hand side and the right hand side of Equation 11 are unchanged. Hence v_j is also undue at the $(s,t)^{th}$ stage and this is also a contradiction.

Suppose that the algorithm moves to a new vertex. Now either $v_{s-1} \in N(\bar{x})_{s-1,t}$ or not. If $v_{s-1} \notin N(\bar{x})_{s-1,t}$, then the previous stage was overdue. This is a contradiction. So suppose that $v_{s-1} \in N(\bar{x})_{s-1,t}$. If the chosen edge is of type (1) or (2) or (3), then the right hand side of Equation 10 goes down by 1 while the right hand side is unchanged. Hence v_j is due at the $(s,t)^{th}$ stage and this is a contradiction. If the chosen edge is of type (4) from Ee_{v_j}, then both left hand and right hand sides of Equation 10 goes down by 1. Hence v_j is normal at the $(s,t)^{th}$ stage and this is a contradiction. If the chosen edge is of type (4) from Ee_{v_i} with $i \neq j$, then left hand stays the same and right hand side of Equation 10 goes down by 1. Hence v_j is normal at the $(s,t)^{th}$ stage and this is a contradiction. The same argument holds if v_1 is overdue. □

Edges of types (1) and (4) are *safe* edges while edges of types (2) and (3) are *risky* edges. Thus, the basic intuition is that our Algorithm aims at con-

structing a simple k-partite graph has to avoid risky edges as much as possible. Now, let $\hat{N}(\bar{x})_s$ be the set of vertices $v_{q,\bar{x}}$ where $q < s$ and whose second index is not x. Observe that if the degree of the $v_{s,x}$, the vertex being inserted, is 2 units greater than $|\hat{N}(\bar{x})_s| - 2$, then BiasedConstructKpartiteRealization() must take preventive measures so that the stage inserting the first edge of v_s is not critical. Otherwise, it will be impossible to insert all the edges of v_s without conceding too many risky edges of type (2). This intuition prompts the following definitions.

A vertex v_r is the *red* vertex of BiasedConstructKpartiteRealization() if v_r is the first vertex during the insertion of which BiasedConstructKpartiteRealization() can reach a critical stage. That is, if BiasedAddvertex() can choose as many risky edges of type (2) as possible, without making previously inserted vertex overdue, then the first critical stage occurs during the insertion of vertex v_r. A vertex v_f is a *fat* vertex if $a_f \geq \hat{N}(\bar{x})_f + 2$. That is, the degree of v_f is greater by at least 2 than the number of vertices v_j with $j < f$ and that can concede an edge to v_f. If this is the case, then since by Condition (1) v_f can be conceded only $\hat{N}(\bar{x})_f$ edges of types (3) or (4), BiasedAddvertex() will be forced to conceded $a_f - \hat{N}(\bar{x})_f \geq 2$ loops. Thus, the insertion of v_f is likely to lead to a spoilt stage if a critical stage is reached before inserting all the edges of v_f. The sequence of vertices $(v_r, v_{r+1}, \ldots, v_z)$ is the *red-fat sequence* of BiasedConstructKpartiteRealization() if v_r is red and v_z is fat. Now, given a red-fat sequence, BiasedConstructKpartiteRealization() has to take preventive measures so that a critical stage is not reached before the last fat vertex is inserted.

Another observation is that if BiasedConstructKpartiteRealization() is inserting vertex v_s that is fat, then as above, Condition (1) imposes that the vertex $v_{s,x}$ can be connected to at most $\hat{N}(\bar{x})_s$ vertices $v_{j,x}$ with $j < s$ and $j \neq 1$. Thus, it is easy to see that the maximal number edges of type (4), denoted by $|E4|_{st}$, which BiasedAddvertex() may concede, from the insertion of the t^{th} edge of vertex v_s until the insertion of the last edge of the last fat vertex v_z is

$$|E4|_{st} = \hat{N}(\bar{x})_s + \hat{N}(\bar{x})_{s+1} + \ldots + \hat{N}(\bar{x})_z.$$

These observations prompt the following definitions. Suppose there is a red-fat sequence $RF = (v_r, v_{r+1}, \ldots, v_z)$, $v_s \in RF$ and $E_{v_s} = a_s - \hat{N}(\bar{x})_s$. Then v_s is *fat-critical* if

$$|Ee|_{st} - |E4|_{st} - (z - s) = a_{z+1} + \ldots + a_n. \tag{12}$$

The $(s,t)^{th}$ stage is *fat-spoilt* if

$$|Ee|_{st} - |E4|_{st} - (z - s) > a_{z+1} + ... + a_n. \tag{13}$$

To make sense of Equation 12, observe that its left hand side is equal to $|Ee|_{z,a_z}$ if BiasedAddvertex() chose the maximal possible number of edges of type (4) (and, conversely, the minimal number of loops). Indeed, an edge $e \in Ee_{z,a_z}$ only if $e \in Ee_{s,t}$. Now, starting from $e \in Ee_{s,t}$, to get the number of edges that are still in the set of excess edges at the $(z, a_z)^{th}$ stage, one has to remove the edges of type (4) which are conceded and thus become unavailable, and there are at most $|E4|_{st}$ of them. Moreover, since $a_s \geq a_{s+1} \geq \cdots \geq a_z$, all these vertices are fat. Thus to each such vertex $v_{j,x}$, v_1 must concede $a_j - \hat{N}(\bar{x})_j$ loops. But, if p loops are conceded to v_j, then $p - 1$ excess-edges (v_1, v_j) are constructed. Hence for every vertex v_j inserted between v_s and v_z, there is one excess edge lost. Hence the term $z - s$ has to be subtracted. Hence the left hand side of Equation 12 is indeed equal to $|Ee|_{z,a_z}$ if the maximal possible number of edges of type (4) is conceded. Using this fact, it is easy to observe that Equation 12 means that the $(s, t)^{th}$ stage is not critical, but if Algorithm BiasedAddvertex() chooses the maximal number of edges of type (4), then the first critical stage would occur after inserting the last edge of the last fat vertex v_z.

Lemma 9 *(i) If the* $(s, t)^{th}$ *stage is fat-critical, then the next stage is fat-critical or fat-spoilt.*

(ii) If the $(s, t)^{th}$ *stage is fat-spoilt, then some future stage is spoilt.*

Proof. (i) First, observe that whatever the choice of edge, the right hand side of Equation 12 stays the same. If BiasedAddvertex() chose an edge of type (4), then on the left hand side, $|Ee|_{st}$ will go down by 1 and $|E4|_{st}$ will also go down by 1, while $z - s$ will stay the same. Hence the next stage is fat-critical. Suppose a loop is chosen, then $|Ee|_{st}$ will stay the same as one loop is lost but an edge of type (4) is created, $|E4|_{st}$ will go down by 1 as v_s can not be connected to the maximal number of edges of type (4) while $z - s$ will stay the same. Hence the next stage is also fat-spoilt.

(ii) The left hand side of Equation 13 is equal to $|Ee|_{z,a_z}$. Hence, if the $(st)^{th}$ stage is fat-spoilt, the $(z, a_z)^{th}$ stage is spoilt. □

While Lemmas 8 and 9 say that once the random walk on $\mathcal{T}$ takes a wrong path, it is impossible to mend it. The next routine gives the preventive measure to avoid getting into that wrong path in the first place. If the algorithm is inserting the t^{th} edge of vertex $v_{s,y}$, then an edge e is available when e is a loop (incident on v_1) or $e \in E_{v_{r,x}}$ with $r < s$ and $\delta(x, y) = 0$.

Routine ChooseCorrectEdge()[Routine choosing edges that lead to a simple graph]

Suppose that BiasedConstructKpartiteRealization() is at its $(s,t)^{th}$ stage. That is, it is inserting the t^{th} edge of vertex $v_{s,y}$. Then

(1) For each vertex $v_{r,x}$ with $r < s$ and $\delta(x,y) = 0$, choose at most one edge from E_{v_r}.

(2) If the stage is normal but not due, choose any available edge uniformly at random.

(3) If the stage is critical but not due nor ripe, choose any available edge of type (1) or (4) uniformly at random.

(4) For $j > s$, if the vertex v_j is due, pick an edge from E_{v_j}. If many such vertices are due, pick an edge uniformly at random from the vertices that are due.

(5) If the stage is critical and ripe, pick a loop.

(6) If v_s, the vertex being inserted, is due, then pick any available edge of type (3) or (4) uniformly at random.

(7) If the stage is fat-critical, then pick any available edge of type (4) uniformly at random.

We illustrate the Routine ChooseCorrectEdge() in Figure 4. Before proving that this algorithm is necessary and sufficient to sample a simple k-partite graph at random, we observe that it runs in $\mathcal{O}(n^2)$ steps, where n is the number of vertices in any realizations of A. Indeed, BiasedConstructKpartiteRealization() calls BiasedAddVertex() n times and BiasedAddVertex() calls ChooseCorrectEdge() a_i times to insert all the edges of vertex v_i. At the i^{th} iteration of BiasedConstructKpartiteRealization(), ChooseCorrectEdge() has to check Equations 1, 7 and 12 once each. Moreover, it has to check Equation 3 for at most $i-1$ vertices. Hence throughout the running of BiasedConstructKpartiteRealization(), ChooseCorrectEdge() has to perform at most $3n + \frac{(n-1)(n-2)}{2}$ checks.

Theorem 10 *Algorithm BiasedConstructKpartiteRealization() reconstructs a simple* k*-partite graph if and only if BiasedAddVertex() calls the routine ChooseCorrectEdge(). In other words, BiasedConstructKpartiteRealization() outputs a simple* k*-partite graph if and only if the choice of edges satisfies conditions (1)–(7).*

The following lemma is required in the proof of Theorem 10.

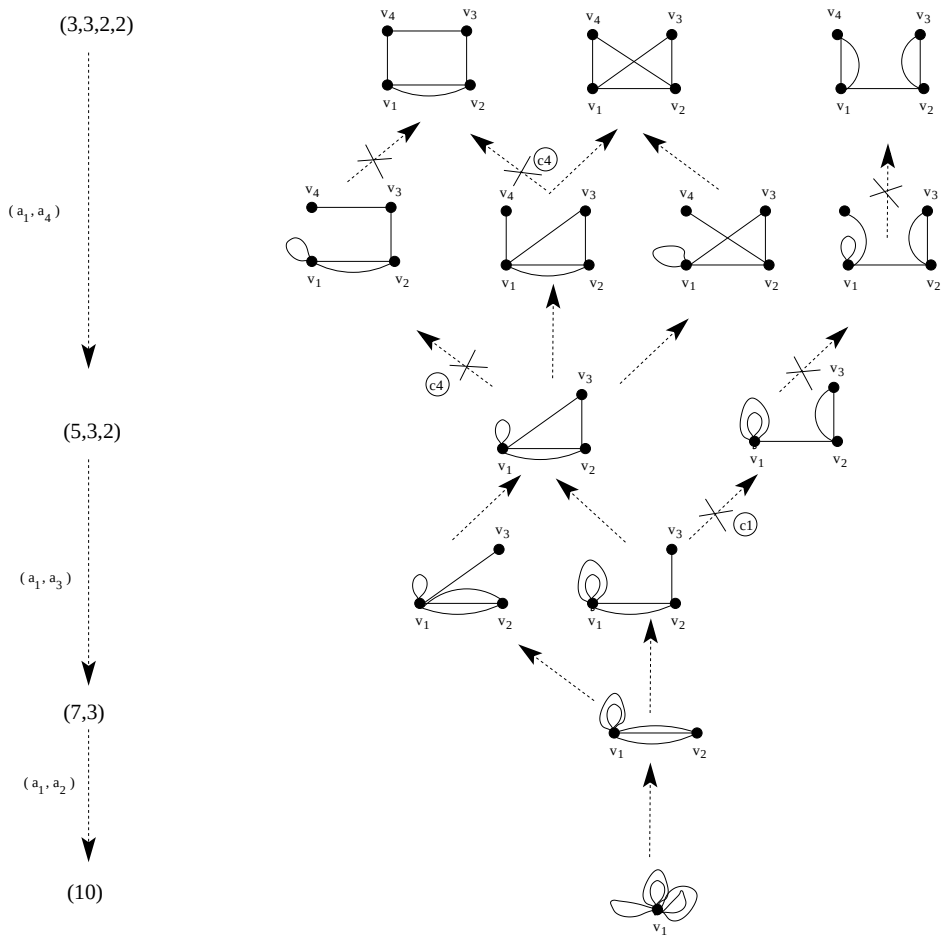

Figure 4: Random reconstruction tree of $(3,3,2,2)$ for simple graph. It is similar to that in Figure 3, but some choice of edges are forbidden. Forbidden moves are marked by a cross and the condition that they fail to satisfy. For example, c1 means condition 1 of the Routine ChooseCorrectEdge()

Lemma 11 *If A is a degree sequence of a simple k-partite graph having n vertices, then the $(2,1)^{the}$ stage of Algorithm BiasedConstructKpartiteRealization() is neither critical nor spoilt nor overdue nor overipe.*

Proof. Before the insertion of vertex v_2, there are m loops incident to v_1 where $m = \frac{\sum_{i=1}^{n} a_i}{2}$. That is,

$$|Ee|_{2,1} = \frac{\sum_{i=1}^{n} a_i}{2}. \tag{14}$$

To prove the result, we only need to show that $|Ee|_{2,1} \leq \sum_{i=2}^{n} a_i$. That is, the number of excess edges is not greater than the number of edges left to insert.

But by Erdos-Gallai criterion, we have

$$a_1 \leq \sum_{i=2}^{n} a_i. \tag{15}$$

Rearranging Equation 14 and plugging Equation 15 into it, we get that

$$2|Ee|_{2,1} = \sum_{i=1}^{n} a_i \leq 2(\sum_{i=2}^{n} a_i). \tag{16}$$

Hence the first stage is not spoilt and vacuously it is not overdue. □

Proof.[Proof of Theorem 10] Suppose for a contradiction that conditions (1)–(7) hold at all the stages but BiasedConstructKpartiteRealization() outputs a k-partite graph G with multiple edges or loops. By condition (1) there can not be a multiple edge connecting two vertices $v_{r,x}$ and $v_{s,y}$ with $r < s$. Moreover, the algorithm would prevent any edge $v_{r,x}$ and $v_{s,x}$ with $r < s$ unless $r = 1$. Hence if G fails to be a simple graph, it must have either a loop or a multiple edge incident to v_1.

So, suppose that in G the vertex v_1 is incident to either a loop e or a multiple edge. Thus, the stage inserting the last edge of the last vertex v_n is either spoilt, or critical and overipe or overdue.

If the last stage is spoilt (overdue), then by Lemma 8 (e, f, g), the previous stage was either spoilt (overdue) or critical (due). If it were spoilt (overdue), then the one prior to it was spoilt (overdue) or critical (due), and so on. Thus by induction, the first stage of BiasedConstructKpartiteRealization() was either spoilt (overdue) or critical (due). This contradicts Lemma 11. Hence some stage later than the $(3,1)^{th}$ must have been critical (due). Thus BiasedConstructKpartiteRealization() must have gone through a series of normal (undue) stages, then a series of critical (due), then a (possible) series of spoilt (overdue) stages prior to the stage inserting the last edge.

So, let the first spoilt (overdue) stage be the $(q,p)^{th}$ stage. So the stage preceding it was critical (due). But, by condition (4), (condition (5)) Algorithm BiasedAddvertex() must have chosen a safe edge so that the $(q,p)^{th}$ stage should be critical (due) by Lemma 8. This is a contradiction.

Suppose that the last stage is critical and overipe. Then using an argument similar to the case where the last stage is spoilt, we also get a contradiction.

Conversely, suppose that some condition (1)–(7) is not satisfied at the $(s,t)^{th}$ stage and let G be the realization output at the end of BiasedConstructKpartiteRealization(). If condition (1) is not satisfied at the $(s,t)^{th}$ stage, then this would create a double edge (v_j, v_s) with $j, s \neq 1$. In that case, the edge stops to be available, and since Algorithm BiasedAddvertex() can not concede it anymore, the double edge would appear in G. Hence G would not be simple.

If condition (3) is not satisfied at the $(s,t)^{th}$, then by Lemma 8(a) and (b) any future stage is spoilt. Hence G is not a simple graph. Suppose that condition (4) is not satisfied. That is, there is a vertex v_j that is due at the $(s,t)^{th}$ stage but Algorithm BiasedAddvertex() does not pick any of the elements of Ee_{v_j} for all the remaining edges conceded to v_s. Then v_j is overdue at the insertion of vertex v_{s+1}, and by Lemma 8(c) it remains overdue until the end of the Algorihm. Hence G is not simple.

Suppose that condition (5) is not satisfied at the $(s,t)^{th}$ stage that is critical. That is, Ee_{v_1}, the set of loops incident to v_1 is ripe but Algorithm BiasedAddvertex() does not pick a loop. Then, by Lemma 8(d), any future stage is spoilt. Hence G is not simple.

Suppose that condition (6) is not satisfied at the $(s,t)^{th}$ stage, that is, the vertex v_s is due and Algorithm BiasedAddvertex() picks a loop. Then v_s will become overdue and G will exhibit a multiple edge (v_1, v_s). If condition (7) is not satisfied, then by Lemma 9 (2) Algorithm ConstructRealization() reaches a spoilt stage. □

Let a *correct edge* be an edge chosen by Algorithm ChooseCorrectEdge. So, if BiasedConstructKpartiteRealization() terminates, we have shown that it always outputs a simple graph. It remains to show that it always terminates by showing that there is always a correct edge so that conditions (4)–(7) can be satisfied.

Theorem 12 *Algorithm ChooseCorrectEdge() always terminates. That is, conditions (4)–(7) can always be satisfied.*

The proof that is quite involved is left to the end of the paper. Still, we have to show that every simple realization can be reached. The next result is instrumental in showing that every simple realization of A can be reconstructed by BiasedConstructKpartiteRealization() if conditions (1)–(7) are satisfied.

Lemma 13 *Let* $G = G_{n_1,n_2,\ldots,n_k}$ *be the* $n_1, n_2, \ldots, n_k$*- complete* k*-partite graph. That is, the* k*-partite graph where the* i^{th} *part contains* n_i *vertices each having degree* $\sum_{j\neq i} n_j$*. Then BiasedConstructKpartiteRealization() satisfying conditions (1)–(7) can reconstruct* G *as a realization of* $A = (A_1 : A_2 : \ldots : A_k)$*, where* A_i *has* n_i *entries equal to* $\sum_{j\neq i} n_j$*.*

Proof. At the second iteration, the algorithm inserts the vertex $v_{2,x}$ by conceding $\sum_{j\neq i} n_j$. It is routine to check that $v_{2,x}$ is due. Now suppose that $v_3 = v_{3,y}$. If $\delta(x, y) = 1$, then v_1 will concede $\sum_{j\neq i} n_j$ loops to v_3. If $\delta(x, y) = 0$, then by Condition (4), v_2 will concede one edge to v_3 and v_1 will have to concede $\sum_{j\neq i} n_j - 1$ loops to v_3. In both cases v_3 is due and by Lemma 8 v_2 is also due. Now, for an induction, suppose that the algorithm is inserting the vertex $v_{s,z}$ and all the preceding vertices are due. Recall that $\hat{N}(\bar{z})_i$ is the set of vertices inserted before $v_{i,z}$ and whose second index in not z. Then v_i will be conceded $|\hat{N}(\bar{z})_i|$ edges from vertices in $\hat{N}(\bar{z})_i$ and $\sum_{j\neq z} n_j - |\hat{N}(\bar{z})_i|$ loops from v_1. Hence v_i will also be due and all the vertices preceding it will be due by Lemma 8. Now it is routine to check that at the n^{th} every vertex is incident to a single available edge. Thus, each of them will concede it and Algorithm BiasedConstructBipartiteRealization() outputs the graph G. □

Let G be a graph, a *delete-minor* of $G' = G\backslash e$ is the graph obtained from G by deleting the edge e. If $A = (A_1 : A_2 : \ldots : A_k)$ is a k-partite degree sequence, let A' be the degree sequence obtained from A by subtracting 1 from two of its entries a_r and a_s, where $a_r \in A_i$ and $a_r \in A_j$ with $i \neq j$. Thus, if A is the degree sequence of a k-partite graph G, then A' is the degree sequence of some delete-minor of G.

Lemma 14 *If BiasedConstructKpartiteRealization() satisfying conditions (1)–(7) can reconstruct* G *as a realization of* A*, then it can reconstruct all the delete-minors of* G *that are realizations of* A'*.*

Proof. Let G be a k-partite graph output by Algorithm BiasedConstructKpartiteRealization() and let $G\backslash e$ be a delete-minor of G. Suppose in the graph G, the edge e is incident to vertices $v_{r,x}$ and $v_{s,y}$ having respectively degrees a_r and a_s, and where $r < s$ and $\delta(x, y) = 0$. Thus in $G\backslash e$, vertices v_r and v_s have degrees $a_r - 1$ and $a_s - 1$. Let f be any edge of $G\backslash e$. Now, since G is output by BiasedConstructKpartiteRealization(), then there is a series of choices of correct edges such that f can be inserted. Now, in that series of choices either e is inserted before f or after. If e is inserted after f, then the same series of choices would insert f in $G\backslash e$. If e is inserted before f, then the same series of

choices, minus the insertion of e, will also lead to the insertion of f in $G\backslash e$, since Algorithm BiasedConstructKpartiteRealization() does not need to insert any edge incident to v_r and v_s as their degrees are down by 1. □

Corollary 15 *Let G be a simple k-partite realization of a degree sequence $A = (A_1 : A_2 : ... : A_k)$, where A_i has n_i entries. Then there is a positive probability that G is output by Algorithm BiasedConstructKpartiteRealization() if conditions (1)–(2) are satisfied.*

Proof. Every simple k-partite graph whose j^{th} part contains n_j vertices can be obtained from $G_{n_1,n_2,...,n_k}$ by a series of deletions. □

3.1 Sampling simple realizations uniformly

The calling of BiasedAddVertex() by BiasedConstructKpartiteRealization() allows to sample all k-partite realizations of A with equal probability. But to construct simple k-partite realizations only, the choice of edges is dictated by the routine ChooseCorrectEdge(). We recall that correct edges are those edges chosen by the routine ChooseCorrectEdge(). It is easy to check that the number of correct edges is not constant across all the graphs on the same level of $\mathcal{T}$. This remark prompts to modify the routine BiasedAddVertex() as follows. A vertex $v_{r,x}$, where $r < s$ and $\delta(x, y) = 0$, is said to be *correctly-adjacent* to v_l at the $(s, t)^{th}$ stage if v_r is connected to v_l by a correct edge at that stage.

SimpleBiasedAddvertex() (A modification of Algorithm Addvertex() to get a uniform distribution on the set of simple k-partite realizations, dividing by the number of vertices inserted up to the s^{th} iteration)
Step 1 To the graph G^l add an isolated vertex called v_{l+1}. Let a_{l+1} be the number of edges to concede to v_{l+1} and let $t = 0$.
Step 2 If v_{l+1} is incident to a_{l+1} edges, return G^{l+1}. Else,
Step 3 Let b^t_{l+1} be the number of different vertices incident to v_{l+1} after the insertion of its t^{th} edge, with $t \geq 1$. If $t = 0$, let b^t_{l+1} be the number of different vertices adjacent to v_l.
Step 4 Choose vertex v_r uniformly at random amongst all the vertices that are correctly-adjacent to v_l.
Step 5 If $e \in E_{v_r}$, then concede e to v_{l+1} with probability $\frac{|V'(G^s)|}{|V(G^s)|.b^t_{l+1}}$, where $V(G^s)$ and $V'(G^s)$ are respectively the sets of vertices inserted up to the s^{th}

iteration and the set of vertices in $V(G^s)$ that are correctly-adjacent to v_1. Increment t by 1 and go back to Step 2.

Theorem 16 *For the degree sequence* $A = (A_1 : A_2 : \ldots : A_k)$*, where* A_i *has* n_i *entries such that* $n_1 + n_2 + \ldots + n_k = n$*, BiasedConstructKpartiteRealization() reaches every simple* k*-partite realization of* A *uniformly at random with probability.*

Proof. Corrolary 15 shows that all simple realizations can be reached by the BiasedConstructKpartiteRealization() if the choice of edges is dictated by the routine ChooseCorrectEdge(). The proof for uniformity is similar to that of Theorem 7.

□

We now give the overall Algorithm and its running time.

Algorithm UniformGenerateSimpleKpartiteRealization()
Input: k-partite degree sequence $A = (A_1 : A_2 : \ldots : A_k)$ where A_i has n_i entries.
Output: A random k-partite realization of A.
Step 1 Put A in non increasing order.
Step 2 Construct the recursion chain of A by calling the routine ConstructKpartiteRecursionChain().
Step 3 Construct a random k-partite realization of A by calling BiasedConstructSimpleKpartiteRealization().

Now, it is known that Step 1 takes $\log(n)$ iterations and as shown earlier Step 2 takes n iterations. In Step 3, ChooseCorrectEdge() does $\frac{n(n-1)}{2}$ checks altogether. Finally, BiasedAddVertex() needs $2m$ iterations to insert all the vertices. Thus, the overall running time is at most

$$\log(n) + \frac{n(n-1)}{2} + 2m \leq n^2 + 2m.$$

3.2 Proof that ChooseCorrectEdge() always terminates successfully

Recall that an edge is *correct* at the $(s,t)^{th}$ stage if ChooseCorrectEdge() may choose it at the $(s,t)^{th}$ stage. We have shown that if BiasedConstructKpartiteRealization() terminates, it always outputs a simple graph. It remains to

show that it always terminates by showing that there is always a correct edge so that conditions (4)–(7) can be satisfied.

Proof.[Proof of Theorem 12]

The proof is by induction on the stage where BiasedConstructKpartiteRealization() is and consists of many lemmas, each dealing with one of the conditions. Obviously all the conditions are satisfied at the $(2,1)^{th}$ stage. Suppose they hold up to the $(s,t-1)^{th}$ stage and let BiasedConstructKpartiteRealization() be at its $(s,t)^{th}$ stage, where the vertex $v_{s,y}$ is being inserted. The next lemma shows that condition (4) is always met. Recall that a safe edge is an edge of type (1) or type (4). □

Lemma 17 *Let the Algorithm BiasedConstructKpartiteRealization() satisfy conditions (1)–(7) and the* $(s,t)^{th}$ *stage is unripe, undue and critical. Then it is always possible to concede a safe edge.*

Proof. Suppose the $(s,t)^{th}$ is critical, but there is no edge of type (1) or type (4). That is, $v_{s,y}$ is already adjacent to v_1 and all vertices $v_{r,x}$, such that $r < s$ and $\delta(x,y) = 0$, have no correct edge.

(a) Suppose v_1 and all the vertices $v_{r,x}$ such that $r < s$ and $\delta(x,y) = 0$ are connected to v_s. Then there is no safe edge if $a_s > \hat{N}(\bar{x})_s$. Thus the vertex v_s is fat. But by condition (7), BiasedConstructKpartiteRealization() can not reach a critical stage before inserting all the edges of v_s. This is a contradiction.

(b) Now suppose there is one vertex, $v_{q,x}$ with $q < s$ and $\delta(x,y) = 0$, that is not connected to $v_{s,y}$. If $|E_{v_q}| > 1$, then there is a correct edge in E_{v_q}. This is a contradiction. So let $|E_{v_q}| \leq 1$. We need the following fact.

Fact 18 *Let* $A = (a_1, a_2, \ldots, a_s, \ldots a_n)$ *be a degree sequence, let* G *be a simple realization of* A *and let* $\hat{G}$ *be the simple graph obtained from* G *by deleting the vertex* v_q *and all edges incident to* v_q*. Then* $\hat{G}$ *is a simple realization of the degree sequence* $\hat{A}$ *obtained from* A *by removing the entry* a_q *and subtracting 1 to all entries* a_i *such that* v_i *is adjacent to* v_q *in* G*. Hence* $\hat{A}$ *is the degree sequence of a simple graph. We call* $\hat{A}$ *a q-reduction of* A*.*

Now, for the degree sequence A, let $P_A = (e_1, e_2, \ldots, e_r)$ be the sequence of correct edge choices leading the critical stage (s,t), where some preceding vertices are not connected to v_s and let v_q be such a vertex. Then $P_{\hat{A}}$ obtained from P_A by removing all the edges incident to v_q is obviously a sequence of correct edge choices for $\hat{A}$ and leading to a critical stage. They are correct, since if e_i is of type (4) or (1) in P_A, it is still of type (4) or (1) in $P_{\hat{A}}$ as

removing edges incident to v_q will leave at least one other edge parallel to it. They lead to a critical stage, since removing edges incident to v_q does not affect the number of excess edges, as v_q is connected to v_1 by 1 edge at most. But according to part (a) the sequence $P_{\hat{A}}$ leads to a critical stage satisfying condition (4). That is, there is a edge e that is correct. Hence the same edge e is correct for A. Thus, we have proved that condition (4) is satisfied in case (b).

□

The next lemma shows that condition (5) is always met.

Lemma 19 *Suppose that BiasedConstructKpartiteRealization() satisfies conditions (1)–(7) and at the* $(s,t)^{th}$ *stage, the vertex* $v_{r,x}$ *is due. Then it is always possible to concede an edge from* E_{v_r}.

Proof. Let a_r denote the degree of the vertex v_r. The proof uses induction on s, the number of vertices already inserted. For a contradiction, we suppose that v_r is due but SimpleBiasedAddvertex() can not pick an edge from E_{v_r}. This is possible only if there are too many vertices that are due. For $s = 1$, this is vacuously not possible. Now suppose for all the stages up to the $(s-1)^{th}$ stage, the result holds, so that none of these stages is spoilt or overdue. In particular, at the insertion of vertex $v_{q,y}$, where q is the greatest index less than s whose second index is y, all the vertices that were due conceded one edge. Suppose at the insertion of vertex v_s, the number of sets that are due is greater than a_s.

(i) Let $v_s = |\hat{N}(\bar{y})|$ be the number of vertices inserted before $v_{s,y}$ and whose second index is not y. Suppose v_1 is not due and that there are $v_s - p$ (for $p \geq 2$) vertices v_j, with $j \neq 1$, that are due at the insertion of vertex v_q. By hypothesis, all the $v_s - p$ concede an edge to vertex v_q and by Lemma 8(c) all these $v_s - p$ vertices are due at the insertion of vertex v_s. Hence, by hypothesis, $a_s < v_s - p$.

If the $(s,t)^{th}$ stage is normal or critical then we have

$$|Ee|_{st} \leq a_s - t + a_{s+1} + \ldots + a_n. \tag{17}$$

But, at $t = 1$, that is, at the insertion of the first edge of vertex v_s, there are at least $v_s - p$ vertices v_j that are due. Thus, on one hand, we have

$$a_s - t + a_{s+1} + \ldots + a_n < (v_s - p)(n - s + 1). \tag{18}$$

Equation 18 follows from the fact $a_s < \nu_s - p$, as we assume that there are too many due vertices, and $a_u \leq a_s$ for $u > s$, and there are $n-s+1$ vertices left to insert.

On the other hand, recalling that E_{ν_1} denotes the set of loops incident to ν_1, we have

$$\begin{aligned} |Ee|_{st} &= |E_{\nu_1}| + |Ee_{\nu_2}| + \cdots + |Ee_{\nu_{s-1}}| \\ &= |E_{\nu_1}| + |Ee|_{due} + |Ee|_{undue} \\ &= |E_{\nu_1}| + |Ee|_{undue} + (\nu_s - p)(n-s+1), \end{aligned}$$

where $|Ee|_{due}$ and $|Ee|_{undue}$ denote the set of excess-edges from vertices that are due and from vertices that are not due respectively, and $|Ee|_{due} = (\nu_s - p)(n-s+1)$, since there are at least $\nu_s - p$ vertices that are due and each has $n-s+1$ excess-edges as there are $n-s+1$ vertices waiting to be inserted.

Hence, if the stage is normal or critical, we have

$$|E_{\nu_1}| + |Ee|_{undue} + (\nu_s - p)(n-s+1) \leq a_s - t + a_{s+1} + \ldots + a_n < (\nu_s - p)(n-s+1). \tag{19}$$

This is impossible. If the stage is spoilt, then SimpleBiasedAddvertex() chose an edge of type (2) or (3) at the $(s-1)^{th}$ stage. This contradicts the inductive hypothesis.

(ii) Suppose ν_1 is due at the $(s,t)^{th}$ stage, so that there are too many vertices that are due and one of them is ν_1. Thus, we have

$$|E_{\nu_1}| + |Ee|_{notdue} + (\nu_s - p)(n-s+1) \leq a_s - t + a_{s+1} + \ldots + a_n \leq (\nu_s - p)(n-s+1). \tag{20}$$

Here we have $\leq$ as $a_s = (\nu_s - p)$, where $\nu_s - p$ is the number of due vertices other than ν_1. But this is possible only if $|E_{\nu_1}| = 0$ and this is a contradiction. □

The next lemma shows that Condition (6) is always met.

Lemma 20 *Suppose that Algorithm BiasedConstructRealization() satisfies conditions (1)–(7) and the $(s,t)^{th}$ stage is critical and ripe. Then it is always possible to concede a loop from E_{ν_1}.*

Proof. The proof uses induction on i, the number of vertices already inserted. The lemma holds vacuously for $i = 1$. Suppose now it holds for all $i < s$. At the insertion of the s^{th} vertex, it may not hold only if there are also at least $\nu_s - p > a_s$ other vertices that are due. Suppose the stage is normal. In that case, as in the proof of Lemma 19, we have

$$|E_{\nu_1}|+|Ee|_{notdue}+(\nu_s-p)(n-k+1) \leq a_s-t+a_{s+1}+...+a_n \leq (\nu_s-p)(n-s+1). \tag{21}$$

But this is possible only if $|E_{\nu_1}| = 0$ and this is a contradiction. Similarly, there is a contradiction if the stage is critical as $|E_{\nu_1}|$ must also be null. Moreover the inductive hypothesis is contradicted if the stage is spoilt.

□

The next lemma shows that condition (7) is always met.

Lemma 21 *Suppose that Algorithm BiasedConstructRealization() satisfies conditions (1)–(7) and at its* $(s,t)^{th}$ *stage* ν_s*, the vertex being inserted, is due. Then there is an available edge that is not a loop.*

Proof. Suppose after the insertion of the $(\tau_1+\tau_2)^{th}$ edge of vertex ν_s, (where τ_2 edges are of type (3) or (4)) the vertex ν_s is due, but $\tau_1+\tau_2 < a_s$. That is, $|Ee_{\nu_s}| = |N(\bar{x})|_s$ but ν_s is not completely inserted. If for some $r \neq 1$ and $r < s$ the vertex $\nu_{r,x}$, with $\delta(x,y) = 0$, is not adjacent to ν_s and $E_{\nu_r} \neq \phi$, then SimpleBiasedAddvertex() picks an edge from E_{ν_r}.

If not, suppose all the vertices $\nu_{r,x}$, with $r < s$ and $\delta(x,y) = 0$ and that are not adjacent to ν_s, we have $E_{\nu_j} = \phi$. Now either (i) the stage is critical or (ii), it is not.

If (i) and all the available edges are loops, then the stage is critical and overipe and this contradicts the inductive hypothesis. Thus, some are loops and some are multiple edges incident to $\nu_{r,x}$ with $r \neq 1$, $\delta(x,y) = 0$, and ν_r is adjacent to ν_s. But, then there must be a vertex ν_j not incident to ν_s with $a_j < a_s$. Let there be p vertices adjacent to ν_s and q vertices ν_j not adjacent to ν_s such that $E_{\nu_j} = \phi$. Then, as assumed above, we have $t + p < a_s$ as ν_s is not fully inserted. But ν_j must be adjacent to some of the vertices ν_i that are adjacent to ν_s. Thus $a_j \leq p < a_s$. This contradicts the fact that $a_1 \geq a_2 \geq ... \geq a_n$.

Now, let all previously inserted vertices ν_j be connected to ν_s. It is easy to see that there must be $s-2$ such vertices which are not ν_1. Thus $a_s = n - s + 1 + (s-2) = n - 1$. But, by Erdös-Gallai criterion, we also have

$a_s \leq n-1$. Hence the vertex v_s is already completely inserted . This is a contradiction. □

The next lemma shows that condition (7) is always met.

Lemma 22 *Suppose that Algorithm BiasedConstructRealization() satisfies conditions (1)–(7) and that the* $(s,t)^{th}$ *is fat-critical. Then there is an available edge of type (4).*

Proof. Suppose, for a contradiction, the $(s,t)^{th}$ is fat-critical, but there is no edge of type (4). That is, all available edges are loops or type (3). That is, for all vertices $v_{r,x}$ such that $r < s$ and $\delta(x,y) = 0$ we have that $a_{r,x} < |\hat{N}(\bar{x})|_r + |N(\bar{x})|_r$. But we know, for all r with $r < s$, $a_s \leq a_r$ and $a_{s,y} > |\hat{N}(\bar{y})|_s$ as v_s is fat. This is a contradiction. □

Thus, we have proved that conditions (4)–(7) are always satisfied at all stages of the running of Algorithm BiasedConstructRealization(). Hence it always terminates reaching a leaf of $\mathcal{T}$ that is a simple graph.

References

[1] M. Bayati, J. H. Kim and A. Saberi, A sequential algorithm for generating random graphs, *Algorithmica* **58** (2010) 860–910. ⇒184, 185, 186

[2] E. A. Bender and E. R. Canfield, The assymptotic number of labelled graphs with given degree sequence, *J. Combin. Theory, Ser A.* **24**, 3 (1978) 296–307. ⇒184, 185

[3] J. Blitzstein and P. Diaconis, A sequential importance sampling algorithm for generating random graphs with prescribed degree sequence, *Internet Math.* **6**, 4 (2011) 489–522. ⇒184, 185, 186

[4] B. Bollobas, A probalistic proof of an assymptotic formula for the number of labelled regular graphs, *European J. Combin.* **1**, 4 (1980) 311–316. ⇒184, 185

[5] R. A. Brualdi, Matrices of zeroes and ones with fixed row and column sum vectors, *Linear Algebra Appl.* **33** (1980) 159–231. ⇒184

[6] T. Brylawsky and J. Oxley, *The Tutte polynomial and its applications*, in N. White, ed., Matroid Applications, Encyclopedia of Mathematics and its Applications, Cambridge University Press, (1992) 123–225. ⇒185

[7] Y. Chen, P. Diaconis, S. Holmes and J. S. Liu, Sequential Monte Carlo methods for statistical analysis of tables, *J. Am. Stat. Assoc.* **100** (2005) 109–120. ⇒184

[8] G. W. Cobb and Y. Chen, An application of Markov Chains Monte Carlo to community ecology, *Amer. Math. Month.* **110** (2003) 265–288. ⇒184

[9] C. Cooper, M. Dyer and C. Greenhill, Sampling regular graphs and Peer-to-Peer network, *Combinatorics, Probability and Computing* **16** (2007) 557–594. ⇒184

[10] P. Diaconis and A. Gangolli, Rectangular arrays with fixed margins, In Discrete Probability and Algorithms (Minneapolis, MN, 1993), *IMA Vol. Math. Appl.* **72** 15–41 , New York Springer, 1995. ⇒184

[11] P. Diaconnis and B. Sturmfels, Algebraic algorithms for sampling from conditional distributions, *Ann Statist.* **26**, 11 (1998) 363–397. ⇒184

[12] P. Erdös and T. G. Gallai, Graphs with prescribed degrees of vertices (Hungarian), *Mat. Lapok* **11** (1960) 264–274. ⇒184, 185

[13] M. Jerrum and A. Sinclair, Approximating the permanent, *SIAM J. Comput.* **18**, 6 (1989) 1149–1178. ⇒184

[14] R. Kannan, P. Tetali and S. Vempala, Simple Markov-chain algorithms for generating bipartite graphs and tournaments, *Random Struct. Algorithms* **14**, 4 (1999) 293–308. ⇒184, 185

[15] K. K. Kayibi, M. A. Khan and S. Pirzada, Rejection sampling of bipartite graphs with given degree sequence, *Acta Univ. Sapientia, Mathematica* **10,** 2 (2018) 249–275. ⇒183, 184

[16] M. Luby, D. Randall and A. Sinclair, Markov chain algorithms for planar lattice structures, *SIAM J. Comput.* **31**, 1 (2001) 167–192. ⇒184

[17] I. Miklós, P. L. Erdös and L. Soukup, Towards random uniform sampling of bipartite graphs with given degree sequence, Preprint. ⇒184, 185

[18] M. E. J. Newman, A. L. Barabasi and D. J. Watts, *The structure and dynamics of networks* (Princeton Studies in Complexity, Princeton UP) (2006) pp 624. ⇒184

[19] S. Pirzada, *An Introduction to Graph Theory*, Universities Press, Hyderabad, India, 2012. ⇒185

[20] V. V. Vazirani, *Approximation algorithms*, Springer-Verlag, Berlin, Heildelberg, New York, 2003. ⇒184

[21] N. Wormald, Models of random regular graphs, In Surveys in Combonatorics (Canterbury), Cambridge University Press, *London, Math. Soc. Lecture Note Ser.* **267** (1999) 239–298. ⇒184, 185

Received: October 22, 2018 • Revised: November 21, 2018