Acta Univ. Sapientiae, Mathematica, **10**, 2 (2018) 249–275

DOI: 10.2478/ausm-2018-0020

# Rejection sampling of bipartite graphs with given degree sequence

Koko K. Kayibi
Department of Physics and Mathematics,
University of Hull, UK
email: kokokayibi@hull.ac.uk

U. Samee
Department of Mathematics,
Islamia College for Science and
Commerce, Srinagar, India
email: pzsamee@yahoo.co.in

S. Pirzada
Department of Mathematics,
University of Kashmir, Srinagar, India
email: pirzadasd@kashmiruniversity.ac.in

Mohammad Ali Khan
Department of Mathematics,
University of Lethbridge, Canada
email: ma.khan@uleth.ca

**Abstract.** Let $A = (a_1, a_2, ..., a_n)$ be a degree sequence of a simple bipartite graph. We present an algorithm that takes $A$ as input, and outputs a simple bipartite realization of $A$, without stalling. The running time of the algorithm is $\ominus(n_1 n_2)$, where $n_i$ is the number of vertices in the part $i$ of the bipartite graph. Then we couple the generation algorithm with a rejection sampling scheme to generate a simple realization of $A$ uniformly at random. The best algorithm we know is the implicit one due to Bayati, Kim and Saberi (2010) that has a running time of $\mathcal{O}(ma_{max})$, where $m = \frac{1}{2}\sum_{i=1}^{n} a_i$ and $a_{max}$ is the maximum of the degrees, but does not sample uniformly. Similarly, the algorithm presented by Chen et al. (2005) does not sample uniformly, but nearly uniformly. The realization of $A$ output by our algorithm may be a start point for the edge-swapping Markov Chains pioneered by Brualdi (1980) and Kannan et al.(1999).

**2010 Mathematics Subject Classification:** 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 graph $G(V(G), E(G))$ is said to be *bipartite* if its vertex set $V(G)$ can be partitioned into two different sets $V_1(G)$ and $V_2(G)$ with $V(G) = V_1(G) \cup V_2(G)$ such that $uv \in E$ if $u \in V_1$ and $v \in V_2$. The graphs considered can have possible parallel edges and loops unless otherwise stated. The *Degree Sequence Problem* is to find some or all graphs with a given degree sequence [30, 34]. More detailed analysis of the Degree Sequence Problem and its relevance can be found in [29]. It is much researched upon for its relevance in network modelling in Ecology, Social Sciences, chemical compounds and biochemical networks in the cell. Especially, ecological occurrence matrices, such as the Darwin finches tables, are $(0, 1)$ matrices whose rows are indexed by species of animals and columns are islands, and the $(i, j)$ entry is $1$ if animal $i$ lives in island $j$, and is $0$ otherwise. Moreover the row sums and columns sums are fixed by field observation of these islands. These occurrence matrices are thus bipartite graphs $G$ with a fixed degree sequence in which $V_1(G)$ is the set of animals and $V_2(G)$ is the set of islands. Researchers in Ecology [8, 9, 15, 31] are highly interested in sampling easily and uniformly ecological occurrence tables, so that by using Monte Carlo methods, they can approximate test statistics to prove or disprove some null hypothesis about competitions amongst animals. Several algorithms are known to sample random realizations of degree sequences, and each one of them has its strengths and limitations. Most of these use Monte Carlo Markov chain methods based on edge-swapping [6, 9, 10, 11, 12, 13, 18, 22, 21, 24]. Since to start a Markov chain still requires to have a realisation of the degree sequence $A$, many algorithms are proposed that generate such a realisation [1, 3, 5, 2, 36]. Most of these algorithms are based on random matching methods. In particular, algorithms proposed in [1, 3, 8] are based on inserting edges sequentially according to some probability scheme. The basic ideas of the algorithm presented in the present paper can be seen as implementing a "dual sequential method", as it inserts sequentially vertices instead of edges.

In the theory of the Tutte polynomial, there are two operations, deletion and contraction, that are dual to each other, see [7] for more details on this topic. Let $G$ be a graph having $n$ vertices and $m$ edges. In $G$, the operation of deleting an edge $e = (v_i, v_j)$ means removing the edge $e$ and the graph thus obtained, denoted by $G \backslash e$, is a graph on $n$ vertices and $m-1$ edges where both the degrees of vertices $v_i$ and $v_j$ decrease by 1. The operation of contracting the graph $G$ by $e = (v_i, v_j)$ consists of deleting the edge $e$ and identifying the vertices $v_i$ 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 easily be shown that random matching methods used in [1, 2, 3, 5, 36] 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 on the stage of the algorithm, 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 has been 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 with 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 a graph with one vertex and $\frac{1}{2}\sum_1^n a_i$ loops, the algorithm re-inserts vertices one by one by choosing the vertices to be joined according to the degrees of the vertices and some probability that depends on the stage of the algorithm. But, to construct a bipartite realization, we force the algorithm to insert first all the vertices in $V_1(G)$ and then all the vertices in $V_2(G)$.

While algorithms that are based on reversing the deletion operation [1, 3] are easy to implement, our algorithm seems more complex as one has to satisfy not only the degree conditions on 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 that 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. In fact, not only the algorithm never restarts, but the algorithm also allows to sample all bipartite realizations with equal probabilities, making their approximate counting much easier than by the importance sampling used in [1, 3]. Better still, this technique can be extended 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 extending the definition of a bipartite degree sequence.

The present paper uses the following notations and terminology. Two edges $e$ and $f$ in $E(G)$ are said to be *multiple* edges if they have the same end vertices (in Matroid Theory, multiple edges are said to be *parallel*). A *simple* bipartite graph is without multiple edges and contains no loops. The *degree* $a_i$ of a vertex $v_i$ is 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 bipartite 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 *bipartite degree sequence*. Thus entries of $A$ can be partitioned as $A_1$ and $A_2$, where $A_i$ denotes the degree sequence of the part $V_i(G)$. We write $V_i$ and $|A_i|$ to denote the set of vertices with degrees in $A_i$ and the sum of entries in $A_i$ respectively . In the sequel, we denote a bipartite degree sequence $A$ as $(A_1 : A_2)$ and the pair $(A_1 : A_2)$ is called a *bipartition* of $A$.

**Remark 1** *If* $A = (A_1 : A_2)$ *is a bipartite degree sequence having* $n$ *entries, and* $A_1$ *and* $A_2$ *have respectively* $n_1$ *and* $n_2$ *entries, then the following are true.*

1. $n_1 + n_2 = n$
2. $|A_1| = |A_2|$.
3. *The maximal entry of* $A_1$ *is less or equal to* $n_2$ *and vice versa.*

Conversely, any partition of entries of $A$ into two sets $B_1$ and $B_2$ satisfying Observation 1 is a bipartition of $A$.

In the sequel, we make use of Rejection Sampling to sample all realizations of the degree sequence with equal probability. Indeed, let $\mathcal{S} = S_1, ..., S_r$ be a set of structures, where $S_i$ is obtained with probability $\pi(S_i)$ such that $\sum_i \pi(S_i) = 1$. That is, the set of $\pi(S_i)$ is a probability distribution function. Let $\min(\pi)$ be the minimal probability amongst all $\pi(S_i)$. The Rejection Sampling scheme consists of generating $S_i$, then accept it with probability $\frac{\min(\pi)}{\pi(S_i)}$ or reject it with probability $1 - \frac{\min(\pi)}{\pi(S_i)}$. It is easy to see that every structure would then be sample with the same probability $\min(\pi)$.

This paper is organized as follows. We first define what is called a recursion chain of a degree sequence, then we present routines for constructing all bipartite realizations. The next section presents criteria and routines to generate simple bipartite realizations only. Then these basic routines are coupled with a rejection sampling routine to get a uniform distribution on the set of all simple bipartite realizations.

# 2 Construction all bipartite 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 and edges of $G$ are labelled $v_1, v_2, \cdots, v_n$. Let $A = (a_1, \cdots, a_n)$ be the degree sequence of $G$, where $a_i$ is 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. We call the sequence $A/(i, j)$ the $(i, j)$-*minor* or simply a *minor* of $A$. The following example illustrates this operation for a bipartite degree sequence.

**Example 1** *Let* $A = (4, 3, 3 : 3, 3, 2, 2)$, *where* $a_1 = 4$, $a_2 = 3$, $a_3 = 3$ *and* $a_4 = 3, a_5 = 3, a_6 = 2, a_7 = 2$. *We have* $A/(1, 2) = (7, 3, 3, 3, 2, 2)$ *and* $A/(4, 2) = (4, 3, 6, 3, 2, 2)$.

Let $A$ be the sequence of integers. $A$ is said to be *graphic* if there is a graph $G$, not necessarily bipartite, 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.

**Theorem 1** *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 the sum of its entries is even, by definition of contraction. 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 other vertices are left unchanged by AddVertex(). Thus $G$ is a realization of $A$. $\square$

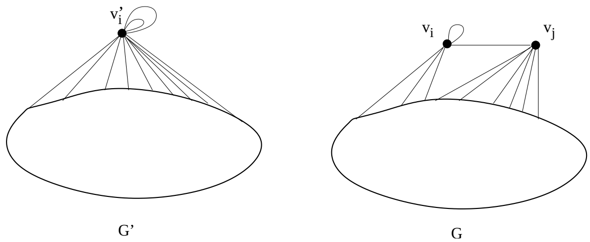


Figure 1: Construction of a graph $G$ from its contract-minor $G''$

To help the 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. See Figure 2 for an illustration.

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 denote by $A^{(i)}$ the node of a recursion chain of a degree sequence $A$, 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 other entries keep the labelling they have before the contraction. In this paper, we consider the recursion chain, called the *accumulating recursion chain*, constructed as follows. Let $A = (A_1 : A_2)$. We order $A = (a_1, a_2, ..., a_n)$ as $(b_1, b_2, ..., b_{n_1} : c_1, c_2, ..., c_{n_2})$, where $A_1 = (b_1, b_2, ..., b_{n_1})$ and $A_2 = (c_1, c_2, ..., c_{n_2})$, such that $b_1 \geq b_2 \geq ... \geq b_{n_1}$ and $c_1 \geq c_2 \geq ... \geq c_{n_2}$ and $n_1 + n_2 = n$. Below is the pseudocode for the recursive construction of the accumulating recursion chain of a bipartite degree sequence.

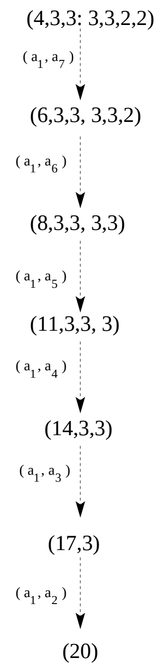


Figure 2: The recursion chain of the bipartition $(4, 3, 3 : 3, 3, 2, 2)$. Nodes of the chain are labelled from $A^{(7)} = A$ to $A^{(1)}$. Notice that we only perform contractions $(v_1, v_{last})$.

**Algorithm ConstructBipartiteRecursionChain()**

Given a bipartite degree sequence $A = (a_1, a_2, ..., a_n) = (b_1, b_2, ..., b_{n_1} : c_1, c_2, ..., c_{n_2})$ with $b_1 \geq b_2 \geq ... \geq b_{n_1}$ and $c_1 \geq c_2 \geq ... \geq c_{n_2}$. 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.

The accumulation recursion chain of $A$ is denoted by $W = (A^{(1)}, A^{(2)}, ..., A^{(n)})$.

The following algorithm generates all the bipartite realizations of $A$. The graph constructed is not necessarily simple. Loosely speaking, this algorithm consists of reversing the recursive process of contraction as implemented by ConstructBipartiteRecursionChain(). This 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 conditions imposed on the choice of edges is that up to the $n_1^{th}$ iteration, only edges $(v_1, v_j)$, with $j \leq n_1$, are constructed. That is, we insert vertices of $V_1$. After the $n_1^{th}$ iteration, only edges $(v_k, v_j)$, with $k > n_1$ and $j \leq n_1$, are constructed. That is, we insert vertices of $V_2$. We call the graphs $G^{(1)}$, $G^{(2)}$,..., $G^{(n)}$ the *partial realizations* of $A$.

**Algorithm ConstructBipartiteRealization()**

Given $W = (A^{(1)}, A^{(2)}, ..., A^{(n)})$, the bipartite accumulating recursion chain of $A$, 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 $i \leq n_1$, AddVertex() only concedes loops. If $i > n_1$ Addvertex() concedes only edges $(v_1, v_j)$ with $1 \leq j \leq n_1$. 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 $a_1$. If we think of recursive contractions of a graph, this is equivalent to saying that the edges are accumulating on $v_1$ as $v_1$ seems to swallow the other vertices one by one. Hence when reversing the contraction operation in ConstructBipartiteRealization(), vertex $v_1$ 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 $v_s$ only if $e$ is incident to $v_1$. This observation prompts the following formal definitions. Let $A = (A_1 : A_2)$ be a bipartite degree sequence, where $A_1$ and $A_2$ have respectively $n_1$ and $n_2$ entries such that $n_1 + n_2 = n$. Up to the $n_1^{th}$ iteration of ConstructBipartiteRealization(), an edge is *available* if it is a loop incident to $v_1$. An edge $e$ is *lost* otherwise. From the $(n_1 + 1)^{th}$ iteration of ConstructBipartiteRealization() onwards, an edge is *available* if it is incident to $v_1$ and a vertex $v_j$ with $1 \leq j \leq n_1$. An edge $e$ is *lost* otherwise. In the obvious way, we say that a vertex is *available* if it is incident to some available edge. Let $V_{av}$, $E_{av}$ and $E_{v_j}$ respectively denote the set of all available vertices, the set of all available edges and the set of available edges that are incident to the vertex $v_j$, for $j \leq n_1$. An edge $e = (v_1, v_j)$ is *conceded* if AddVertex() disconnects it from $v_1$ so that $e$ becomes $e = (v_j, v_k)$ for some vertex $v_k \neq v_1$. We then say that $v_1$ (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 ConstructBipartiteRealization().

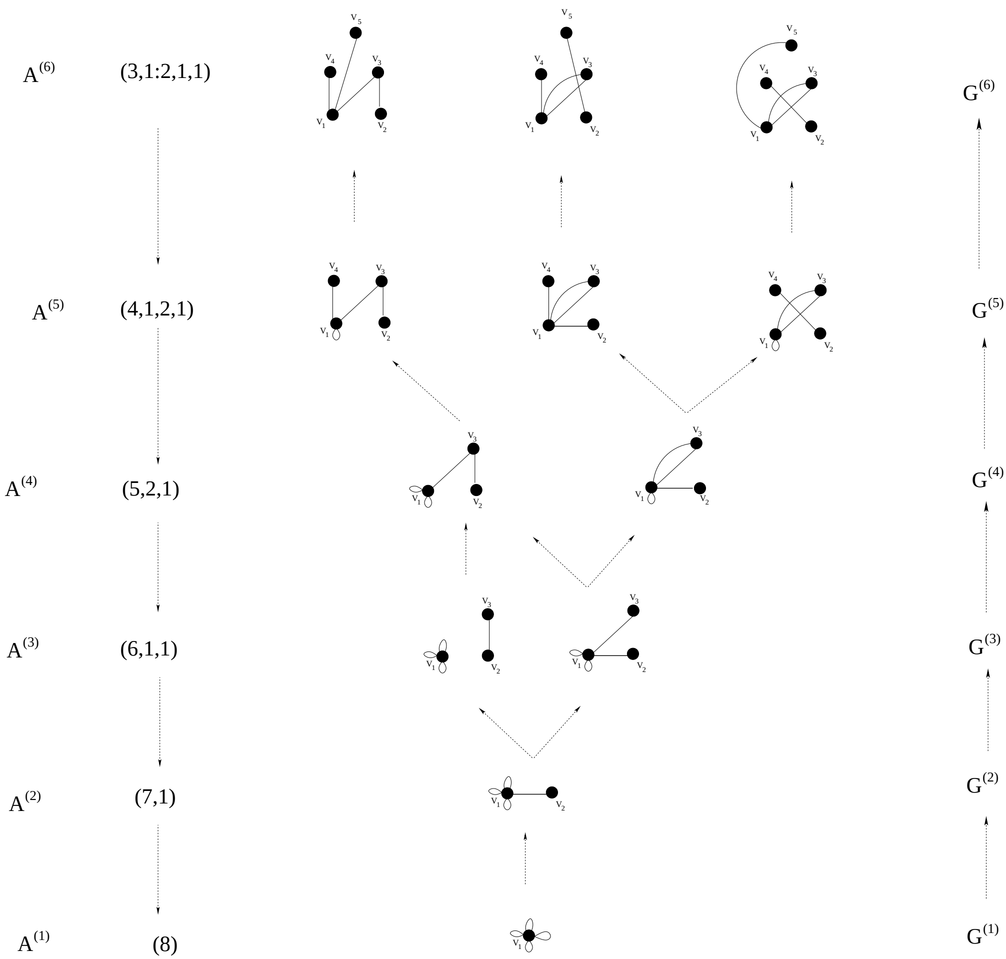


Figure 3: Random reconstruction tree of $(3, 1 : 2, 1, 1)$. Graphs drawn on the same height as the degree sequence $A^{(i)}$ corresponds to all the graphs having $A^{(i)}$ as their degree sequence. Notice that only realizations of $A^{(6)}$ are bipartite.

The next observation is an obvious consequence of the definition of the algorithm ConstructBipartiteRealization(). We single it out for the sake of clarity as it is used in the sequel.

**Remark 2** *From the $(n_1+1)^{th}$ iteration of ConstructBipartiteRealization(), the number of available edges is equal to the number of edges left to be inserted until ConstructBipartiteRealization() terminates.*

It is because the number of available edges at the end of $(n_1)^{th}$ iteration is equal to half the sum of degrees $a_i \in A_1$, and by the definition of the bipartite degree sequence, this number is equal to half the sum of degrees $a_j \in A_2$.

**Theorem 2** *Let $A = (a_i, a_2, \cdots, a_n) = (A_1 : A_2)$ be a bipartite degree sequence having $n$ entries where $A_1$ and $A_2$ respectively have $n_1$ and $n_2$ entries, such that $n_1 + n_2 = n$. Let $W$ be the bipartite recursion chain of $A$. Then Algorithm ConstructBipartiteRealization() constructs in time linear on $m = \frac{a_i+a_2+\cdots+a_n}{2}$ a bipartite graph $G$ having $n$ vertices and $m$ edges such that $G$ is a realization of $A$. Moreover, every bipartite realization of $A$ can be constructed in this way.*

**Proof.** By Algorithm AddVertex(), the graph $G^{(n)}$ output by Algorithm ConstructBipartiteRealization() is assured to be a realization of $A$. We need only to prove that $G^{(n)}$ is bipartite. Now, since up to the $n_1^{th}$ iteration of ConstructBipartiteRealization(), the routine AddVertex() always chooses loops incident to $v_1$, vertices inserted from the second iteration up to the $n_1^{th}$ iteration of ConstructBipartiteRealization() (i.e., vertices in $V_1$) can never be adjacent to each other. Moreover, from the $(n_1+1)^{th}$ to the $n^{th}$ iteration, AddVertex() never chooses an edge $(v_1, v_j)$ with $j > n_1$. Thus all the vertices inserted from the $(n_1+1)^{th}$ iteration onwards (i.e., vertices in $V_2$) are never adjacent to each other. Thus, we only have to show that (in $G^n$), $v_1$ is not adjacent to any vertex inserted before the $n_1^{th}$ iteration of AddVertex(). So, suppose that $G^n$ contains an edge $e = (v_1, v_j)$ with $j \leq n_1$. But, at the beginning of the $(n_1+1)^{th}$ iteration, the number of all edges incident to $v_1$ is equal to the sum of the degrees of the vertices left to insert until the end of the Algorithm. Thus one vertex $v_j$ with $j > n_1$ is not fully inserted. This is a contradiction.

It remains to prove that any bipartite realization $G$ of $A$ can be constructed in this way. So, let $G$ be a realization of $A$ and let $e = (v_i, v_j)$, where $v_i \in V_1$ and $v_j \in V_2$, 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 vertex $v_i$ and $v_j$ respectively were inserted at the $i^{th}$ and $j^{th}$ iteration of ConstructBipartiteRealization(), with $i \leq n_1$ and $j > n_1$. We need to show that at the $j^{th}$ iteration, there is a positive probability to have an edge $e$ that is incident to $v_i$ and $e$ is available. Assume to the contrary, that is, at the $j^{th}$ iteration all the edges incident to $v_i$ must be lost. Now all the edges incident to $v_i$ are lost before that $j^{th}$ iteration only if at some stage of the running of Algorithm ConstructBipartiteRealization(), there are only the edges that are available and these are exhausted before reaching the $j^{th}$ iteration. Thus, at the $j^{th}$ iteration there are no more available edges. That is, there is no edge incident to $v_1$. But this means that $a_{n_1+1} + a_{n_1+2} + \ldots + a_{j-1} \geq m$, contradicting Observation 2.

As for the running time, Algorithm ConstructBipartiteRealization() calls Algorithm AddVertex() once for every new vertex $v_k$ to be inserted. If $v_k$ has degree $a_k$, 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 ConstructBipartiteRealization() is $a_1 + a_2 + \ldots + a_n = 2m$. □

## 3 Construction of simple bipartite graphs

Till now, ConstructBipartiteRealization() generates any bipartite realization of the bipartite degree sequence $A$. But, it is easy to modify AddVertex() so that the output of ConstructBipartiteRealization() is always a simple graph. One obvious condition can be stated as follows.

(a) If the Algorithm is inserting the $j^{th}$ edge of vertex $v_s$ ( with $j > 1$ and $v_s \in V_2$) and $v_k$ ($v_k \in V_1$) is already adjacent to $v_s$, then no more available edge incident to $v_k$ should be chosen. This would prevent ConstructBipartite-Realization() from outputting graphs with multiple edges $(v_s, v_k)$. Thus this condition is necessary, but it 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 $v_k$ so as not to construct multiple edges $(v_s, v_k)$, ConstructBipartiteRealization() may fall into a stage where there are more edges incident to $v_k$ than there are vertices left to insert, and $G$, the graph output by ConstructBipartiteRealization() would then have a multiple edge $(v_1, v_k)$.

(c) Let $A_1$ and $A_2$ be (separatly) ordered in non decreasing order, where $a_1$ is the largest entry of $A_1$ and $a_{n_1+1}$ is the largest entry of $A_2$. Let $M_k$ be the set of the last $k$ entries of $A_1$ and let $\max(k) = a_{n_1-k+1}$. Let there be an entry $a_s$ in $A_2$ satisfying the following.

(f1). $s - n_1 \geq \max(k)$, (i.e., the number of entries of $A_2$ preceding $a_s$ is greater or equal to the maximal entry in $M_k$)

(f2). $a_s > n_1 - k$, (i.e., inserting $v_s$ would require more neighbours than there are vertices in $V_1 \backslash M_k$) and,

f(3).

$$\sum_{j=n_1+1}^{s-1} a_j \geq \sum_{i=k}^{n_1} a_i + \sum_{i=1}^{k-1} \max(0, a_i - n + s).$$

(that is, the number of edges required to insert vertices of $V_2$ prior to $v_s$ exceeds the number of edges available on vertices in $M_k$ plus the minimum number of edges that a vertex $v_i$ (with $v_i \in V_1 \backslash M_k$) has to concede prior to the $s^{th}$ iteration to prevent $v_i$ from having more edges than there are vertices left to be inserted from the $s^{th}$ iteration onwards.)

If $a_s \in A_2$ satisfies (f1), (f2) and (f3), then $a_s$ is said to be *k-fat*. Let $F_k$ denote the set of all the entries that are k-fat. See an illustration in Figure 4.

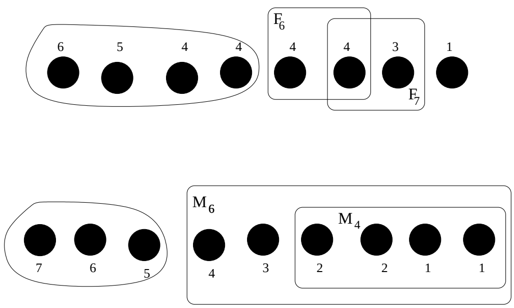


Figure 4: A=(7,6,5,4,3,2,2,1,1 : 6,5,4,4,4,4,3,1), where $A_1 = (7, 6, 5, 4, 3, 2, 2, 1, 1)$ and $A_2 = (6, 5, 4, 4, 4, 4, 3, 1)$. Entries are labelled so that the leftmost entry of $A_1$ is $a_1$ and the rightmost entry of $A_2$ is $a_{17}$. The entries $a_{14}$ and $a_{15}$ are 6-fat while $a_{15}$ and $a_{16}$ are 7-fat.

Now, if (a) is to be respected and ConstructBipartiteRealization() chose every vertex in $M_k$ to concede an edge to every one of the $s - n_1$ vertices preceding $v_s$, then ConstructBipartiteRealization() would get stuck at the stage of inserting vertex $v_s$. This is because by (f1) and (f3), no vertex in $M_k$ would have any edge to concede to $v_s$ and so there would be a maximum of $n_1 - k$ available vertices. But by (f2), vertex $v_s$ needs more adjacent neighbors than the only $n_1 - k$ available vertices. Hence, ConstructBipartiteRealization() must take some precautionary measures by not exhausting all the edges incident to vertices in $M_k$ prior to the insertion of $v_s$.

Figure 5 illustrates how the Algorithm would get stuck at its $s^{th}$ iteration.

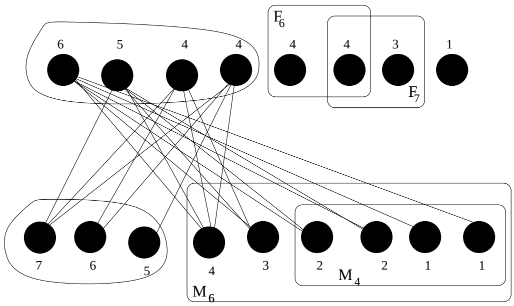


Figure 5: This is a choice of edges that may exhaust all the edges incident to vertices in $M_6$ prior to the $14^{th}$ iteration. In this choice, vertex $v_1, v_2$ and $v_3$ must concede $3, 2$ and $1$ edges respectively lest they would have too many edges after the $13^{th}$ iteration. Still, vertex $v_{14}$ would not get inserted fully and the Algorithm would stall.

Although (a), (b) and (c) 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 lengthy, 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 be inserted at each stage of the Algorithm. Moreover, checking these conditions at each iteration of AddVertex() requires checking $\mathcal{O}(n^2)$ inequalities altogether. Thus it does not add to the running time.

Let $A = (A_1 : A_2)$ be a bipartite degree sequence of a simple graph, where $A_1$ and $A_2$ have respectively $n_1$ and $n_2$ entries such that $n_1 + n_2 = n$. We recall that $E_{av}$ represents the set of available edges. That is, edges that are incident to $v_1$ and vertices inserted before the $n_1^{th}$ iteration of ConstructBipartiteRealization(), that is, the vertices of $V_1$. For $v_j \in V_1$, we recall that $E_{v_j}$ is the set of available edges incident to $v_j$. That is, the set of parallel edges connecting $v_1$ and $v_j$. Obviously $E_{v_j} \subseteq E_{av}$ for all $j$. In particular, $E_{v_1}$ is the set of loops incident to $v_1$.

Some of the Algorithms given in the literature, such as in [1], have the disadvantage that it has to restart. The algorithm given here allows to choose only edges such that it never has to restart. In order to be able to do that, the choice of edges at every stage must be such that no vertex is incident to too many edges of the 'wrong type'.

If at its $s^{th}$ iteration, Algorithm ConstructBipartiteRealization() is inserting the vertex $v_s$ that has degree $a_s$, then ConstructBipartiteRealization() has to *call* the routine AddVertex() that has to go through $a_s$ iterations. We recall that the $(s,t)^{th}$ stage of ConstructBipartiteRealization() is the iteration where AddVertex() inserts the $t^{th}$ edge of the $s^{th}$ vertex. Let $X_{s,t}$ and $|X|_{s,t}$ denote respectively a set and its cardinality at the $(s,t)^{th}$ stage of ConstructBipartiteRealization().

To help the reader, we first introduce the motivation for the definitions. At each stage of constructing a simple graph, every vertex $v_j$, where $v_j \in V_1$, must be connected by at most one edge to any other $v_k$, where $v_k \in V_2$. So, if some vertex $v_j$ has more available edges than the vertices left to be inserted after its $s^{th}$ iteration, ConstructBipartiteRealization() would never be able to get rid of all these multiple edges, which would then appear in the final graph. This prompts the following definitions. The vertex $v_j$ where $j \leq n_1$ ( i.e., $v_j \in V_1$) is *due* if

$$|E_{v_j}|_{st} = n - (s-1), \tag{1}$$

that is, $E_{v_j}$ has as many edges as there are vertices left to be inserted. The vertex $v_j$ is *overdue* if

$$|E_{v_j}|_{st} > n - (s-1), \tag{2}$$

that is, there are too many available edges incident to $v_j$ and whatever are the future choices, the Algorithm would never output a simple graph. The vertex $v_j$ is *undue* if it is neither due nor overdue. Obviously, a stage is *due, undue, overdue* if there is a vertex that is due, undue or overdue,respectively.

Let $M_k$ be the set of the last $k$ entries of $A_1$. An entry $a_s$ in $A_2$ is *k-fat* if conditions (f1), (f2) and (f3) are satisfied. We let $F_k$ to denote the set of vertices that are $k$-fat. A bipartite degree sequence $A$ is *fat* if it contains a $k$-fat entry for some integer $k > 0$.

Let $r_i = a_i - n_1 + k$, where $k$ is the largest integer such that $a_i$ is k-fat. The $(s,t)^{th}$ stage is *ruined* if there is an entry $a_i$ with $i > s$ (that is, the vertex $v_i$ is not inserted yet) that is fat and the number of vertices in $M_k$ that are available is less than $r_i$. It is *not ruined* otherwise.

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

**Lemma 1** *Suppose ConstructBipartiteRealization() is inserting the vertex* $v_s$ *such that* $s > n_1$, *(i.e., inserting* $v_s$ *into* $V_2$). *Then the following hold.*

(a) *If the vertex* $v_j$ *is due, it is due or overdue at the next stage. If it is overdue, it is overdue at any future stage.*

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

(c) *If the* $(s,t)^{th}$ *stage is ruined, then the next stage is also ruined.*

**Proof.**

(a) Suppose $v_j$ is due and Addvertex() does not choose an edge from $E_{v_j}$. Since no edge of $E_{v_j}$ is chosen, the left side of Equation 1 remains same while the right hand side either goes down by one if ConstructBipartiteRealization() moves to a new vertex $v_{s+1}$ or stays the same if ConstructBipartiteRealization() moves to another edge $t+1$ of the same vertex $v_s$. Hence the next stage is due or overdue. On the other hand, if Addvertex() chooses an edge from $E_{v_j}$, the left hand side goes down by 1 and the right one stays the same. But if $E_{v_j}$ concedes only one edge to $v_s$ (as we shall see shortly), $E_{v_j}$ is still due at the insertion of vertex $v_{s+1}$. Similar arithmetical arguments as above show that if $v_j$ is overdue, it stays overdue.

(b) Suppose $v_j$ 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

$$|E_{v_j}| < n - (s-1). \tag{3}$$

Now, either the last edge inserted is chosen from $E_{v_j}$ or not. Moreover, in either case, Algorithm ConstructBipartiteRealization() moves to a new vertex or not. If it stays on the same vertex and the chosen edge is not from $E_{v_j}$, the right and the left hand sides of Equation 3 are both unchanged. Hence $v_j$ is undue at the $(s,t)^{th}$ stage, which is a contradiction. If it stays on the same vertex and the chosen edge is from $E_{v_j}$, the left hand side of Equation 3 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 again is a contradiction.

Suppose ConstructBipartiteRealization() moves to a new vertex. If the chosen edge is not from $E_{v_j}$, the right hand side of Equation 3 goes down by 1 while the right hand side is unchanged. Hence $v_j$ is due at the $(s,t)^{th}$ stage, a contradiction. If the chosen edge is from $E_{v_j}$, both left hand and right hand sides of Equation 3 go down by 1. Hence $v_j$ is normal at the $(s,t)^{th}$ stage, a contradiction.

(c) Assume that the $(s,t)^{th}$ stage is ruined. That is, there is a fat vertex $v_i$ that is not inserted yet, but the number of vertices in $M_k$ which are available is less than $r_i$. But, at the next stage, this number can never increase. Thus it would also be ruined. □

While Lemma 1 says that once ConstructBipartiteRealization() takes a wrong path, it is impossible to mend it, the next routine gives preventive measures to avoid getting into that wrong path in the first place.

**ChooseCorrectEdge()**

Let $A$ be not fat and ConstructBipartiteRealization() is at its $(s,t)^{th}$ stage with $s > n_1$ (that is, inserting vertex $v_s$ into $V_2$). Then,

(1) for each vertex $v_j \in V_1$, do not choose an edge in $E_{v_j}$ if there is already an edge $(v_s, v_j)$.

(2) if the vertex $v_j$ is due, pick an edge from $E_{v_j}$. If many vertices are due, pick an edge uniformly at random from the vertices that are due.

Now assume that $A$ is fat and for some integer $k > 0$, $F_k$ is not empty. Then, for every entry $a_i \in F_k$ choose at random $r_i = a_i - n_1 + k$ different entries in $M_k$. The only condition imposed on the choice is that an entry $a_j$ can be chosen at most once for each fat vertex and at most $a_j$ times for all the fat vertices combined. If $a_i$ is k-fat, let $R_i$, called the *reserve pool* of $a_i$, be the set of vertices in $M_k$ chosen for $a_i$. Let $R_{ij}$, the *reserve matrix*, be an $n_1$ by $n_2$ matrix whose columns are indexed from $1$ to $n_1$ (indices of entries of $A_1$), and rows are indexed from $n_1 + 1$ to $n$ (indices of entries of $A_2$), and $R_{ij} = 1$ if the entry $a_j \in R_i$, and zero otherwise. Obviously, the sum of entries in row $i$ is equal to $r_i$ and the sum of entries of column $j$ must be less or equal to $a_j$. At the $(s,t)^{th}$ stage, a vertex $v_j \in V_1$ is *exhausted* if the sum of row $j$ plus the number of vertices adjacent to $v_j$ equals $a_j$. (that is, the number of edges already conceded by $v_j$ and the number of edges of $v_j$ in the reserve pools equals $a_j$).

(3) If ConstructBipartiteRealization() is at its $(s,t)^{th}$ stage with $s > n_1$ and $a_s$ is not fat, then apply (1) and (2) subject to not choosing a vertex $v_j$ if $v_j$ is exhausted. If $a_s$ is fat, first choose all the vertices in $R_s$, then apply (1) and (2) if necessary.

## Complexity Issues

Before proving that the conditions set in routine ChooseCorrectEdge() are necessary and sufficient to sample a simple bipartite graph at random, we observe that, if $A = (a_1, a_2, ..., a_n) = (A_1 : A_2)$ where $A_1$ and $A_2$ have respectively $n_1$ and $n_2$ entries such that $n_1 + n_2 = n$ and $\sum_{i=1}^{n} a_i = 2m$, ChooseCorrectEdge() runs altogether in $\mathcal{O}(n_1 n_2)$ steps. Indeed, at the $s^{th}$ iteration of ConstructBipartiteRealization(), ChooseCorrectEdge() has to check Equation 1 only once for every vertex $v_j \in V_1$. But there are $n_2$ iterations and $n_1$ vertices $v_j$ with $j \le n_1$. This takes $\mathcal{O}(n_1 n_2)$ steps. Constructing the Reserve Matrix $R$ requires $\mathcal{O}(n_1 n_2)$ steps as one has to check Conditions (f1), (f2) and (f3) for each of the $n_2$ entries of $A_2$ and writing the $n_1 n_2$ entries of the matrix $R$.

**Theorem 3** *Algorithm ConstructBipartiteRealization() reconstructs a simple graph if and only if AddVertex() calls the routine ChooseCorrectEdge(). In other words, ConstructBipartiteRealization() outputs a simple graph if and only if the choice of edges satisfies Conditions (1), (2) and (3).*

**Proof.** Assume to the contrary that Conditions (1) and (2) hold but ConstructBipartiteRealization() outputs a bipartite graph $G$ with multiple edges or loops. By Condition (1) there can not be a multiple edge connecting two vertices $v_j$ and $v_k$ such that $j \le n_1$ and $k > n_1$. Moreover, by the definition of the routine ConstructBipartiteRealization(), there can not be a double edge $(v_k, v_l)$ where $k, l > n_1$. Hence if $G$ fails to be a simple graph, it must have either a loop or a multiple edge incident to $v_1$ and $v_j$ such that $j \le n_1$.

So, in $G$, let the vertex $v_1$ is incident to either a loop $e$ or a multiple edge $(v_1, v_j)$ such that $j \le n_1$. But, by the definition of the bipartition, the number of edges incident to $v_1$ at the end of the $n_1^{\text{th}}$ iteration of ConstructBipartite-Realization() equals the number of edges left to be inserted until Construct-BipartiteRealization() terminates. Hence, some vertex $v_k$ such that $k > n_1$ is not fully inserted. This is a contradiction.

Conversely, let the condition (1) or (2) be not satisfied and let $G$ be the realization output by ConstructBipartiteRealization(). If condition (1) is not satisfied at the $(s,t)^{\text{th}}$ stage, this would create a double edge $(v_j, v_s)$ with $j \le n_1$ and $s > n_1$. Now, since Algorithm Addvertex() can not concede the double edge $(v_j, v_s)$ anymore as they are lost, the double edge $(v_j, v_s)$ would appear in $G$. Hence $G$ would not be simple. Assume that the condition (2) is not satisfied. That is, there is a vertex $v_j$ with $j \le n_1$ that is due at the $(s,t)^{\text{th}}$ stage, where $s > n_1$, but Algorithm Addvertex() does not pick any of the elements of $E_{v_j}$ 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 1(b) it remains overdue until the end of Algorihm 2. Hence $G$ is not simple as it must have a multiple edge $(v_i, v_j)$. If condition (3) is not satisfied, Algorithm ConstructBipartite-Realization() may stall. □

Let a *correct edge* and *vertex* be an edge chosen by Algorithm ChooseCorrectEdge and a vertex incident to a correct edge, respectively. So if Construct-BipartiteRealization() 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 (1) and (2) can be satisfied at every stage of ConstructBipartiteRealization().

**Theorem 4** *Algorithm ConstructBipartiteRealization() always terminates. That is, Conditions (1) and (2) are always satisfied at every stage of ConstructBipartiteRealization().*

**Proof.** Suppose $A$ does not contain any fat entry. That is, as long as an edge $e = (v_1, v_j)$ is a correct vertex, it can be chosen. Obviously, Condition (1) can always be forced on AddVertex(). But, while trying hard to satisfy Condition (1), the algorithm may let a vertex $v_j$ of $V_1$, to become overdue. If at the $(s,t)^{th}$ stage the vertex $v_j$ is due, we prove that it is always possible to concede an edge from $E_{v_j}$ to $v_s$.

So assume to the contrary that $v_j$ is due but Addvertex() can not pick an edge from $E_{v_j}$. This is possible only if there are too many vertices that are due. That is, $a_s < n_1' \leq n_1$, where $n_1'$ is the number of vertices that are due at the $(s,t)^{th}$ stage. But we also have $a_s \geq a_{s+1} \geq ... \geq a_n$. Moreover, as all these $n_1'$ vertices are due, each of them is incident to $n - s$ available edges. Hence we have $a_s + a_{s+1} + \cdots + a_n < n_1'(n - s)$. That is, there are more available edges than there are edges left to be inserted until ConstructBipartiteRealization() terminates. This contradicts Observation 2.

Let the entry $a_i$ be k-fat. If all the correct edges $e = (v_1, v_j)$ such that $v_j \in M_k$ are conceded prior to the insertion of the vertex $v_i$, then by definition of fat entry, Algorithm ConstructBipartiteRealization() would stall as there would not be enough edges to connect to $v_i$. But, we assume that the Algorithm reserved $r_i$ edges to concede to $v_i$. Hence $v_i$ can always be inserted. So, we only need to check (c1), whether putting some edges in reserve would prevent some non-fat vertex $v_s$ from being inserted for lack of correct edges and, (c2), whether it is always possible to construct the reserve matrix $R_{ij}$.

(c1) Assume that $s < i$. That is, $v_s$ precedes $v_i$. Let all vertices preceding $v_s$ have been inserted but there are not enough correct edges to insert $v_s$. This is possible if reserving edges for vertices in $F_k$ and inserting vertices preceding $v_s$ exhausts $q$ vertices of $V_1$ and $a_s > n_1 - q$. Without loss of generality, we may assume that the last $q$ vertices of $V_1$ are exhausted. So, let the available vertices be vertices $v_1, \ldots, v_{n_1-q+1}$. If the number of available edges is less than $a_s$, then $A_1 < A_2$. This is a contradiction. So, let the number of available edges be greater or equal to $a_s$. Thus the number of available vertices is less than $a_s$, so that Condition (1) prevents $a_s$ edges from being connected to $v_s$. Let $H$

be the graph obtained after the insertion of $v_{s-1}$ by 'fully' connecting all the vertices in $V_2\backslash v_s$, making sure to connect vertices in $F_k$ with edges that are reserved for them in $R_{ij}$. Then, by the definition of $r_i$, it is easy to check that every vertex in $F_k$ is adjacent to every vertex in $V_1\backslash M_k$. Also, since all the vertices in $M_q$ are exhausted after the insertion of $v_{s-1}$, one can check that none of the vertices in $M_q\backslash M_k$ is adjacent to a vertex in $V_2\backslash(F_k \cup V_{\leq s})$, where $V_{\leq s}$ denotes the set of vertices from $v_{n_1+1}$ up to $v_s$. ( i.e., $V_2\backslash(F_k \cup V_{\leq s})$ is the set of vertices between $v_s$ and $F_k$). Thus, only the vertices in $V_1\backslash M_q$ are adjacent to vertices in $V_2\backslash(F_k \cup V_{\leq s})$. Since all the vertices, except for $v_s$ are properly connected and $|A_1| = |A_2|$, the number of available edges is $a_s$ but the number of available vertices is less than $a_s$. Therefore, by the pigeonhole principle, there is an available vertex having at least two available edges. Without loss of generality, we may consider $v_1$ to be the culprit.

Now, since only the vertices in $V_1\backslash M_q$ are adjacent to the vertices in $V_2\backslash(F_k \cup V_{\leq s})$, either $v_1$ is adjacent to all the vertices in $V_2\backslash(F_k \cup V_{\leq s})$ or it is not. If it is, then $v_1$ was due during an iteration prior to or during the insertion of $v_{s-1}$ and the algorithm did not select it to concede an edge. This is a contradiction. Suppose that it is not adjacent to some vertex $v_t \in V_2\backslash(F_k \cup V_{\leq s})$. Then $a_t < n_1 - q$, since $v_t$ is fully connected. But, by the non decreasing ordering of $A_2$, we also have $a_t \geq a_i$. Moreover, since $a_i \in F_k$, we have $a_i \geq n_1 - k$. Hence we have $a_i \geq n_1 - k > n_1 - q > a_t$. This is also a contradiction. Therefore vertex $v_s$ can be fully inserted. See Figure 6 which helps to understand notations in part (c1).

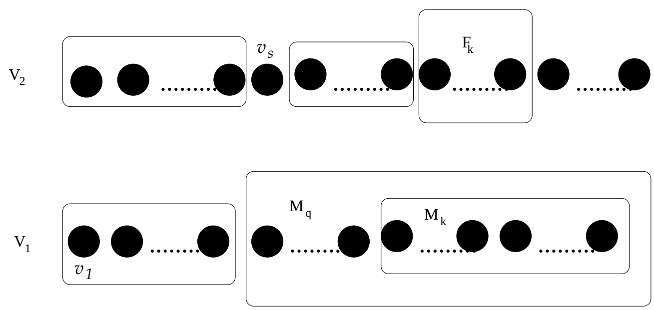


Figure 6:

Finally, let $a_i$ be k-fat, $a_s$ be not k-fat and $s > i$. (that is, $v_s$ is to be inserted after $v_i$). If there are not enough correct edges to connect to $v_s$, then $|A_1| < |A_2|$. This is a contradiction.

(c2) Suppose that it is not possible to built the reserve matrix. But, since

$a_i \le n_1$ for all entries in $F_k$, this would imply either $\sum_{F_k} a_i > \sum_{V_1 \setminus M_k} a_i + \sum_{M_k} a_i = |A_1|$, or $a_i > n_1$ for some entry $a_i \in F_k$. This is a contradiction. □

It still remains to show that the algorithm constructs all the simple realizations of $A$.

**Lemma 2** *Let $G_{n_1,n_2}$ be the $n_1, n_2$- complete bipartite graph. That is, the bipartite graph where one part contains $n_1$ vertices each having degree $n_2$ and the second part contains $n_2$ vertices each of degree $n_1$. Then ConstructBipartiteRealization() satisfying Conditions (1) and (2) can reconstruct $G_{n_1,n_2}$ as a realization of $A = (A_1 : A_2)$ where $A_1$ has $n_1$ entries $a_i = n_2$ and $A_2$ has $n_2$ entries $a_j = n_1$.*

**Proof.** At the beginning of the $(n_1 + 1)^{th}$ iteration, $E_{v_j} = n_1$ for each of the $n_1$ vertices already inserted. Hence each such vertex is due. Now, the vertex $v_{n_1+1}$ has degree $a_{n_1+1} = n_1$ by the definition of $A$. Hence, by Condition (2), AddVertex() chooses one edge from each of the $n_1$ vertices $v_j$ with $j \le n_1$ and inserts $v_{n_1+1}$ completely. By Lemma 1, each $v_j$ is still due at the $(n_1 + 2)^{th}$ iteration. Again, by Condition (2), AddVertex() chooses one edge from each of the $n_1$ vertices $v_j$ with $j \le n_1$ and inserts $v_{n_1+2}$ completely. And so on, until the insertion of vertex $v_n$, and Algorithm ConstructBipartiteRealization() outputs the graph $G_{n_1,n_2}$. □

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 $A = (A_1 : A_2)$ is a bipartite degree sequence, let $A'$ be the degree sequence obtained from $A$ by subtracting 1 from two of its entries $a_i$ and $a_j$, where $a_i \in A_1$ and $a_j \in A_2$. Thus, if $A$ is the degree sequence of a bipartite graph $G$, then $A'$ is the degree sequence of some delete-minor of $G$.

**Lemma 3** *If ConstructBipartiteRealization() satisfying Conditions (1) and (2) 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 bipartite graph output by Algorithm ConstructBipartite-Realization() and let $G\backslash e$ be a delete-minor of $G$. In the graph $G$, let the edge $e$ be incident to vertices $v_j$ and $v_k$ having respectively degrees $a_j$ and $a_k$, where $j \le n_1$ and $k > n_1$. Thus in $G\backslash e$, vertices $v_j$ and $v_k$ have degrees $a_j - 1$ and $a_k - 1$. Let $f$ be any edge of $G\backslash e$. Since $G$ is output by ConstructBipartite-Realization(), there is a series of choices of correct edges such that $f$ can be inserted. In that series of choices either $e$ is inserted before or after $f$. If $e$ is inserted after $f$, the same series of choices would insert $f$ in $G \backslash e$. If $e$ is inserted before $f$, the same series of choices, minus the insertion of $e$, would also lead to the insertion of $f$ in $G \backslash e$, since Algorithm ConstructBipartiteRealization() does not need to insert any edge incident to $v_j$ and $v_k$ as their degrees are down by 1. □

**Corollary 1** *Let $G$ be a simple bipartite realization of a degree sequence $A = (A_1 : A_2)$ where $A_1$ and $A_2$ have $n_1$ and $n_2$ entries respectively. Then there is a positive probability that $G$ is output by Algorithm ConstructBipartiteRealization() if Conditions (1) and (2) are satisfied.*

**Proof.** Every simple bipartite graph having one part of $n_1$ vertices and another of $n_2$ vertices can be obtained from $G_{n_1,n_2}$ by a series of deletions. □

### 3.1 Sampling all bipartite realizations uniformly

Although Theorem 2 shows that the routine ConstructBipartiteRealization() can construct a realization of $A$ in time linear on the number of edges of its realizations, we need the next result to show that it can construct any bipartite realization of $A$ with equal probability, provided we define the probability $\pi(e)$ with which AddVertex() has to insert the edge $e$. If at its $k^{th}$ iteration ConstructBipartiteRealization() is to insert the vertex $v_k$ that has degree $a_k$, then ConstructBipartiteRealization() has to *call* AddVertex() that has to go through $a_k$ iterations. Let the $(s,t)^{th}$ stage of ConstructBipartiteRealization() 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 bipartite 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 7 for an illustration.

**Rejection sampling**

Let $G$ be a realization of $A$. That is, $G$ is a leaf of the tree $\mathcal{T}$. Obviously, there are many paths of $\mathcal{T}$ leading to $G$. Let $p$ be such a path and let $\pi_p(G)$ denote the probability to reach $G$ along the path $p$. Now $\pi_p(G)$ can easily be computed on the fly since $\pi_p(G) = \prod_{e \in E(G)} \pi(e)$, where $E(G)$ denotes the set of edges of $G$ and $\pi(e)$ is the probability to choose the edge $e$. Now $\pi(e) = \frac{1}{|V_{cor}}|$, where $V_{cor}$ is the set of all correct vertices at the insertion of $e$. The only problem is that $G$ can be reached from many paths. The next result proves that all these paths have equal probability.

**Lemma 4** *Let* $G$ *be a realization of* $A$ *that can be reached through the paths* $p$ *and* $q$ *of* $\mathcal{T}$*. Then* $\pi_p(G) = \pi_q(G)$*.*

**Proof.** Let $E(G)$ denote the set of edges of $G$. Then, $p$ can be seen as a re-ordering of a subset of edges chosen along $q$. Now, since the vertices are added in the same order along $q$ as along $p$, we may only consider the case where $p$ and $q$ differ on a single vertex and edges $e$ and $f$ are interchanged in $p$ and $q$. Let $V_{cor}(e)$ and $V_{cor}(f)$ denote the sets of correct vertices at the insertion of $e$ and $f$, respectively. If the Algorithm can choose either the edge $e$ or $f$, then $V_{cor}(e) = V_{cor}(f)$ and the probability to choose either must be the same. □

Lemma 4 allows to compute $\pi(G)$ on the fly. For any path $p$ leading to $G$, we have

$$\pi(G) = \prod_{e \in G} \pi(e) = \prod_{e \in G} \frac{1}{|V_{corr}(e)|},$$

where $V_{corr}(e)$ is the set of vertices in $V_1$ that are incident to some correct edge. Hence, to get $\pi(G)$ on the fly, one set $\pi(G) = \pi(G^{n_1}) = 1$. For every partial realization $G^{(i)}$ from $(G^{n_1})$ to $G$ multiply $\pi(G)$ by $\frac{1}{|V_{corr}(e)|}$. Finally output $\pi(G)$ with $G$. Now let $\min(\pi)$ be a lower bound of the probabilities to reach of the realizations of $A$. This lower bound can be calculated using only parameters of $A$. Indeed, if $|V_{av}(e)|$ stands for the number of vertices in $V_1$ that are adjacent to $v_1$ at the insertion of edge $e$, then we have the inequality $\frac{1}{|V_{av}(e)|} \leq \frac{1}{|V_{corr}(e)|} \leq \pi(e)$ and, for any realization $G$, we have

$$\prod_{e \in G} \frac{1}{|V_{av}(e)|} \leq \prod_{e \in G} \pi(e) \leq \pi(G).$$

Finally, since $|V_{v_1}(e)| \leq n_1$ and every realization of $A$ has $m$ edges, we get

$$\frac{1}{n_1^m} \leq \prod_{e \in G} \frac{1}{|V_{av}(e)|} \leq \prod_{e \in G} \pi(e) \leq \pi(G).$$

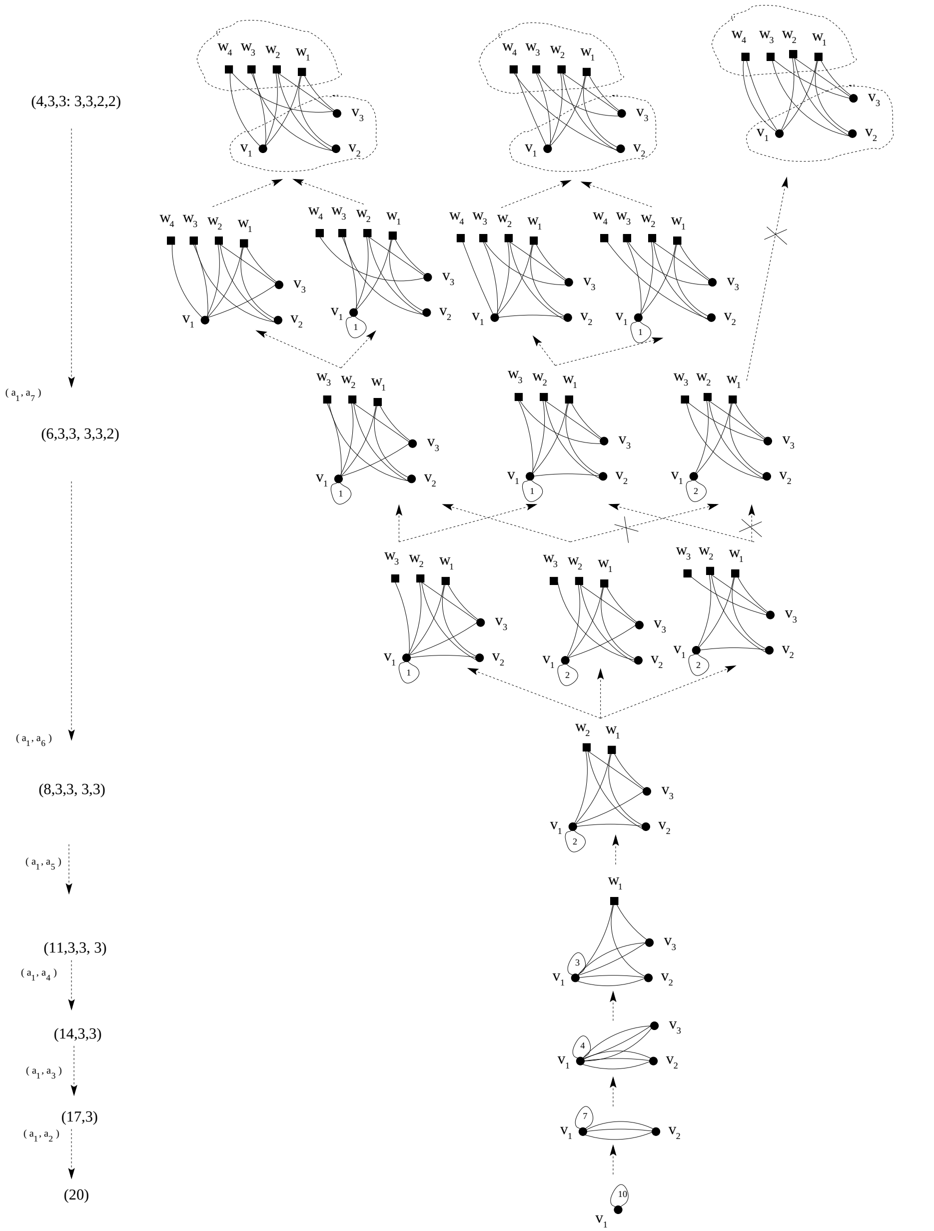


Figure 7: Random reconstruction tree of $(4, 3, 3 : 3, 3, 2, 2)$. The level of $\mathcal{T}$ on the same height as the degree sequence $A^{(i)}$ corresponds to all the graphs having $A^{(i)}$ as their degree sequence. The arrows that are crossed denote the edges that would not lead to a simple realization.

**Algorithm RejectionSampling()**

Input: Bipartite degree sequence $A = (A_1 : A_2)$, where $A_1$ and $A_2$ have $n_1$ and $n_2$ entries respectively such that $n_1 + n_2 = n$ and an integers $r_1$.

Output: A sequence of $r_1$ bipartite simple realizations of $A$ where every realization has equal probability.

*Step 1* Put $A_1$ and $A_2$ in non decreasing order.

*Step 2* Construct the recursion chain of $A$ by calling the routine ConstructBipartiteRecursionChain().

*Step 3* Call ConstructBipartiteRealization() to construct the realization $G$. Let $\pi(G)$ be the probability computed on the fly and get $u$, a random number in $(0, 1)$. If $u < \frac{\min(\pi)}{\pi(G)}$, accept $G$ and go back to Step 3 until one gets $r_1$ realizations. Else, reject $G$ and go back to Step 3 until one gets $r_1$ realizations.

Obviously, Algorithm RejectionSampling() samples every realization of $A$ with the same probability equal to $\min(\pi)$. Now, it is known that Step 1 takes $\log(n_1) + \log(n_2)$ iterations and, as shown earlier, Step 2 takes $n_1 + n_2$ iterations. In Step 3, ChooseCorrectEdge() does $\mathcal{O}(n_1 n_2)$ inequality checks altogether while AddVertex() needs $2m$ iterations to insert all the vertices. Thus, the overall running time to get the minimum probability is given by

$$\log(n_1) + \log(n_2) + r_1(n_1 n_2 + 2m) = \mathcal{O}(r_1(n_1 n_2 + 2m)) \asymp \mathcal{O}(3r_1 m) \asymp \mathcal{O}(m).$$

Finally, $T$, the running time of generating a realization of $A$ uniformly, is a geometric random variable with expected running time given by $\frac{1}{(\pi(acc)}$ where $\pi(acc)$ is the acceptance probability for the realization $G$ with the highest probability of being output by ConstructBipartiteRealization(). So

$$\pi(acc) = \frac{\min(\pi)}{\pi(G)} = \frac{\min(\pi)}{\prod_{e \in G} |V_{corr}(e)|}.$$

Now if $n_2 \to \infty$, then $|V_{corr}(e)| \to \frac{n_1}{2}$ on average. Therefore,

$$\pi(acc) \to \frac{\min(\pi)}{(\frac{2}{n_1})^m} = \frac{\frac{1}{n_1^m}}{(\frac{2}{n_1})^m} = \frac{1}{2^m}.$$

Hence $T \to 2^m$. For the typical Darwin tables $m$ is about 40 edges. Thus $2^m$ is a manageable running time.

## Acknowledgements

The first author would like to express his gratitude to the University of Bristol, the third to University of Kashmir, India, and SERB-DST for continuous support in research.

## References

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

[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.

[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.

[4] F. Boesch and F. Harary, Line removal algorithms for graphs and their degree lists, *IEEE Trans. Circuits Syst. CAS*, **23** (12) (1976), 778–782.

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

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

[7] 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.

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

[9] G. W. Cobb and Y. Chen, An application of Markov Chains Monte Carlo to community ecology, *American Math. Monthly*, **110** (2003), 265–288.

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

[11] 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.

[12] P. Diaconnis and B. Sturmfels, Algebraic Algorithms for sampling from conditional distributions, *Annal. Statist.*, **26** (11) (1998), 363–3977.

[13] P. Erdős and T. G. Gallai, Graphs with prescribed degrees of vertices (Hungarian), *Mat. Lapok*, **11** (1960), 264–274.

[14] P. L. Erdős, I. Miklós and Z. Toroczkai, A simple Havel-Hakimi type algorithm to realize graphical degree sequences of directed graphs, *Electron. J. Combin.*, 17(1) (2010), #R66.

[15] M. Gail and N. Mantel, Counting the number of r x c contingency tables with fixed margins, *J. American Stat. Assoc.*, **72** (1977), 859–862.

[16] G. Guenoche, Counting and selecting at random bipartite graphs with fixed degrees, *Revue francaise d'automatique, d'informatique et de recherche operationelle*, **24** (1) (1990), 1–14.

[17] J. Guillaume and M. Latapy, Bipartite Graphs as models of complex networks, Preprint.

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

[19] M. Jerrum and A. Sinclair, Fast uniform generation of regular graphs, *Theoretical Computer Science*, **73** (1) (1990), 91–100.

[20] M. Jerrum and A. Sinclair, The Markov Chain Monte Carlo method: an approach to approximate counting and integration, in *Approximation Algorithms for NP-hard problems*, (D. S. Hochbaum ed.) PWS Publishing House, 1995, 482–519.

[21] 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.

[22] K. K. Kayibi, S. Pirzada and T. A. Chishti, Sampling contingency tables, *AKCE International Journal of Graphs and Combinatorics*, in press.

[23] H. Kim, Z. Toroczkai, P. L. Erdős, I. Miklós and L. A. Székely, Degree-based graph construction, *J. Phys. A: Math. Theor.*, **42** (2009), 392001.

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

[25] B. McKay and N. C. Wormald, Uniform generation of random regular graphs of moderate degree. *J. Algorithms*, **11** (1) (1990), 52–67.

[26] L. McShine, Random sampling of labeled tournaments, *Electron. J. Combin.*, **7** (2000), #R8.

[27] I. Miklós, P. L. Erdös and L. Soukup, Towards random uniform sampling of bipartite graphs with given degree sequence, *Electron. J. Combin.*, **20, 1** (2013), P16.

[28] M. Molloy and B. Reed, A critical point for random graphs with a given degree sequence, *Random Struct. Algorithms*, **6** (1995), 161–180.

[29] 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.

[30] S. Pirzada, *An introduction to Graph Theory*, Universities Press, Orient-BlackSwan, Hyderabad (2012).

[31] A. Roberts and L. Stone, Island-sharing by archipelago species, *Oecologica*, **83** (1990), 560–567.

[32] H. J. Ryser, Combinatorial properties of matrices of zeros and ones, *Can. J. Math.*, **9** (1957), 371–377.

[33] A. Sinclair, Convergence rates of Monte Carlo experiments, in *Numerical Methods for Polymeric Systems*, (S. G. Whittington ed.) IMA volumes, 639–648.

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

[35] F. Viger and M. Latapy, Efficient and simple generation of random simple connected graphs with prescribed degree sequence, *Lecture Notes in Computer Science* **3595** (2005), 440–449.

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

*Received: March 28, 2018*