Uniform generation of random directed graphs with prescribed degree sequence

Diploma Thesis

Petko P. Fiziev
Department of Mathematics and Computer Science
Institute of Computer Science
Free University of Berlin, Germany

Advisor: Prof. Martin Vingron
Computational Molecular Biology Department
Max Planck Institute for Molecular Genetics, Berlin, Germany

March 31, 2006

CONTENTS

To my father.

1. INTRODUCTION

Directed graphs (digraphs) are a mathematical abstraction for binary relations. One can visualize them on a piece of paper by drawing dots connected with arrows. Each dot represents an object from a given universe of discourse and each arrow means that the two objects that are source and target of it belong as a tuple to the relation. In graph theory the dots are called vertices and the arrows - edges. A directed graph G is defined as the tuple $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of all vertices and $\mathcal{E}$ is the set of all edges in it.

Many problems in fields ranging from economics to molecular biology can be formulated in the terms of graph theory. It is interesting then to analyze the properties of the graphs and draw conclusions about the world that they model. One of the most tempting questions is the one about the origin of the relation. Does it result from a natural evolution-like process or was it designed specifically for the task? Some networks that arise by the hands of humans are designed; probably the best known example are electronic circuits. If displayed correctly one can tell only with a naked eye flip-flops from multiplexers, adders from multipliers and so on in a graph that represents the connections between transistors in a microprocessor chip, if one knows how a single flip-flop or a multiplier looks like. On the other hand, there are many networks that arise by a process that is a mixture between evolution and design like public transport or the internet. Clearly, there is some design involved here. For example, in public transport there are often cycles and long bidirectional lines that cross each other. On the other hand, it is always adapted by shortcuts to things of local importance like banks, governmental buildings, museums, airports and so on.

It is hard to say, whether there will be ever a network that models some human activity and does not show any signs of design. Nevertheless, the same does not necessarily apply to graphs that represent natural processes in life on low level like signal transduction, metabolic pathways, gene regulation, protein-protein interactions, neuron connectivity in brain and many more. It is tempting to say that there was design in them, but to prove it might be far from easy. Research in this direction has started some years ago. A well known example is the discovery that a three-node structure called "feedforward" loop (FFL) occurs statistically more often in the network of transcriptional regulation in the bacteria E.Coli [22]. To conclude this the authors measured the concentrations of all possible subgraphs with three vertices in this network and compared them to the average in a sample from the space of all possible directed graphs that have the same degree sequence. Clearly, this approach is very sensitive to the choice of the null hypothesis and the algorithms that generate the random samples. If one states the background model incorrectly it is very probable that wrong conclusions will be drawn. In this case, it imposes the claim that each transcription factor controls the same amount of genes and is controlled itself by the same number of transcription factors. If this reflects the natural constraints of gene regulation in bacteria is questionable, but seems to be a good starting point.

When the background model is formulated, it is theoretically possible to calculate the estimated frequencies of subgraphs [8, 9] or other features. Unfortunately, to do this exactly is extremely difficult and to our best knowledge there is no scientific work that has this kind of results. Since it is also hard to evaluate how good the approximations are, in practice another approach is applied to test a null hypothesis. One generates randomly representatives from its distribution and estimates from their properties the statistical significance of the real data. This strategy is based on the Monte Carlo Method, first described by Nicholas Metropolis and Stanislaw Ulam in 1949 [13]. While they first applied it to predict the behavior of neutrons in nuclear fission experiments during World War II, the advancing computing technologies made it possible for this method to be used in various other fields where an analytical solution is unknown or complicated enough to work out. Two intuitive conditions must be held for the method to give correct estimations. First, it is necessary to have a procedure that samples uniformly according to the null hypothesis. Inventing one that performs this task correctly could be difficult depending on the complexity of the background model as we shall see soon. Introducing biases could be very misleading in some cases though tolerable in other and deserves a separate analysis. Second, one has to set a reasonable limit for the size of the sample that will be generated. Too few may lead to less accurate conclusions and too many may require more time than one can afford, so this tradeoff has to be resolved adequately. Unfortunately, to estimate analytically the error rate is often as difficult solving the whole problem by this means. Therefore, one is forced to apply some rules of the thumb that sound reasonable.

In our work we will focus on the algorithms for sampling representatives from the background model used for network motif detection. It is organized as follows: Chapter 2, Methods, describes three algorithms for uniform generation of random directed graphs with prescribed degree sequence: the switching, the matching and the ”Go with the Winners” algorithm. We will prove their correctness, try to analyze their performance and also see some interesting examples. Chapter 3, Results, presents their sequential implementation as well as a parallel one of the ”Go with the Winners” algorithm, a comparison of their running times and some empirical examples of biased sampling. In Chapter 4 we discuss the untreated problems in this field and possible future directions of research. Chapter 5, Summary, gives a brief overview of what we have done.

2. METHODS

The analysis of different properties of random graphs is of interest for some couple of decades. The classical work of Paul Erdös and Alfrèd Rènyi of 1959 [6] lays the foundation of the theory, which was further developed by many other authors. An excellent source is also the book of Bèla Bollobàs [3].

First we will start with some definitions and notations that will be used along this chapter. A *digraph* $G = (\mathcal{V}, \mathcal{E})$ is a tuple of two sets. $\mathcal{V}$ is the set of all vertices and we denote their number with n. $\mathcal{E}$ denotes the set of all edges and their number is denoted by E. A digraph G is represented as an adjacency matrix M where $m_{i,j} = 1$ if there exists an edge $(i, j) \in \mathcal{E}$, and 0, otherwise. The restriction that there are no self-edges implies that $m_{i,i} = 0$ for all $i = 1 \dots n$ and we will call these entries *structural zeros*. Alternatively, we will also denote an edge pointing from vertex i to vertex j with (i, j).

A *degree sequence* $D = \langle (in_1, out_1), (in_2, out_2), \dots, (in_n, out_n) \rangle$ is a sequence of n pairs of nonnegative integers (in_i, out_i) for $1 \leq i \leq n$. A degree sequence is called *graphical* if there exist at least one labeled simple directed graph G with labels $1 \dots n$ such that in_i is equal to the number of incoming edges and out_i to the number of outgoing edges at vertex i for all $i = 1 \dots n$. The numbers in_i and out_i are called also *indegree* and *outdegree* of i, respectively. It follows easily that

$$\sum_{i=1}^{n} in_i = \sum_{i=1}^{n} out_i = E$$

and

$$\sum_{j=1}^{n} m_{i,j} = out_i \texttt{ and } \sum_{i=1}^{n} m_{i,j} = in_j.$$

In this manuscript we use and discuss graphical degree sequences only, so when we say degree sequence we assume implicitly that it is graphical. The set of all possible labeled simple directed graphs with degree sequence D, which is discrete and finite, will be denoted by $\mathfrak{G}$ and its cardinality by N.

Now we can state the problem that we try to solve in this work.

Problem. Given a graphical degree sequence, choose uniformly one representative from $\mathfrak{G}$. Alternatively: From the set of all $n \times n$ 0-1 matrices with and zero diagonal, and given row and column sums, pick one at random. These two formulations are equivalent.

The simplest solution, the "brute force" approach, is not acceptable in practice, because the number of possible graphs grows exponentially for some degree sequences [23]. Therefore one prefers a generative procedure that produces random samples of representatives from $\mathfrak{G}$. Such algorithms are used for the background model in the motif detection software *mfinder* [15]. There are three main algorithms: the switching, the matching and "Go with the Winners". In the following sections we will explore their theoretical foundations.

2.1 The Switching Algorithm

The switching algorithm algorithm was studied in several scientific works [10, 15, 20, 27]. It was presented with modifications, not all of which result in uniform sampling, and we shall discuss each of them in detail. First we will make some additional definitions.

Definition 2.1.1. *The* ***Hamming distance*** *between two adjacency matrices A and B is defined as the number of cells for which $a_{i,j} \neq b_{i,j}$ and is denoted by $d(A, B)$.*

It is easy to see that $d(A, B) = d(B, A)$. Also, if A and B have equal row and column sums, $d(A, B)$ should be even. To understand why the latter is true, let us take a row i in A and in B. Without loss of generality, let $a_{i,j} = 1$ and $b_{i,j} = 0$ for some j. Since $\sum_{j=1 \ldots n} a_{i,j} = \sum_{j=1 \ldots n} b_{i,j}$, it means that there must exist an index $k \neq j$ such that $a_{i,k} = 0$ and $b_{i,k} = 1$. In this way one could find for any pair of entries in A and B such that $a_{i,j} = 0$ and $b_{i,j} = 1$ another pair for which $a_{i,k} = 1$ and $b_{i,k} = 0$. This means that the Hamming distance is even.

Furthermore, we state the following

Definition 2.1.2. *An* ***alternating rectangle*** *in M is a set four distinct indices $\{i_1, i_2, j_1, j_2\}$ for which $m_{i_1,j_1} = 1$, $m_{i_1,j_2} = 0$, $m_{i_2,j_2} = 1$ and $m_{i_2,j_1} = 0$.*

We define the operation *switching along an alternating rectangle* in M as the bitwise negation of the four cells of the rectangle. For example, here is the result of switching along the alternating rectangle $\{1, 3, 2, 4\}$ in the following adjacency matrix:

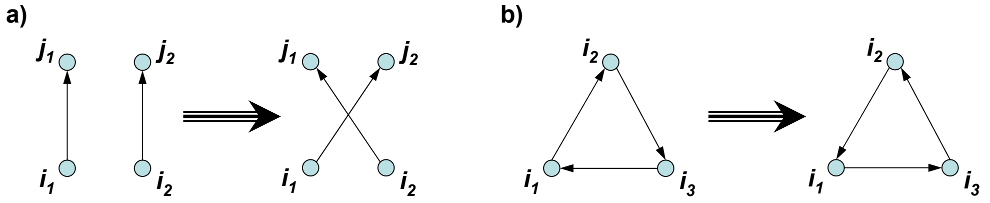

Fig. 2.1: Switch along a) an alternating rectangle and b) an alternating hexagon.

$$\begin{pmatrix} 0 & \mathbf{1} & 0 & \mathbf{0} \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \mathbf{0} & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{0} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

A switch like this is a formal definition of interchanging the ends of the two edges (i_1, j_1) and (i_2, j_2) assuring that no loop or multiple edge will be created (see Figure 2.1a). The obtained graph has the same degree sequence, because the row and column sums remain unchanged. Moreover, after the switch the four cells m_{i_1,j_1}, m_{i_1,j_2}, m_{i_2,j_2} and m_{i_2,j_1} form another alternating rectangle, so repeating the operation again would bring the graph in its initial state. We will continue with a second

Definition 2.1.3. *An **alternating hexagon** in M is a set of three distinct indices $\{i_1, i_2, i_3\}$ for which $m_{i_1,i_2} = 1$, $m_{i_1,i_3} = 0$, $m_{i_2,i_3} = 1$, $m_{i_3,i_2} = 0$, $m_{i_3,i_1} = 1$ and $m_{i_2,i_1} = 0$.*

Analogously we define the operation *switching along an alternating hexagon* as the bitwise "NOT" of its cells. For example, here is the effect of a switch along the alternating hexagon $\{1, 2, 3\}$ in:

$$\begin{pmatrix} 0 & \mathbf{1} & \mathbf{0} & 0 \\ \mathbf{0} & 0 & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \mathbf{0} & \mathbf{1} & 0 \\ \mathbf{1} & 0 & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{1} & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Switching along an alternating hexagon causes a one-way cycle of three nodes to turn around without conflicts (see Figure 2.1b). Again, the operation is reversible and does not change the degree sequence.

It is worth noting that one could not always achieve this effect with a series of switches along alternating rectangles. This is because interchanging the ends of any two edges in a one-way cycle of three nodes would create a self edge. Conversely, the effect of a switch along an alternating rectangle is impossible to be obtained by a series of switches along alternating hexagons. This is because the latter causes each affected edge to interchange its source and target. Therefore, is impossible to interchange the ends of two edges that have distinct source and target vertices. We will explore these properties deeper in the upcoming paragraphs, but before this we will formulate the first version of the switching algorithm:

Algorithm 2.1.1.

1. *Choose an integer $\tau > 0$.*
2. *Generate one directed graph (denoted as G_0) that has the given degree sequence.*
3. *Pick an alternating rectangle or hexagon in the adjacency matrix of G_0 at random and perform a switch along it to produce a graph G_1.*
4. *Do step 3. recursively τ times.*
5. *Output the graph G_τ.*

Intuitively, one would think that G_τ will be a completely random representative from the space $\mathfrak{G}$ if τ is big enough. We will soon see that this is not exactly correct, but let us first take a closer look at the algorithm in this form. It represents a Markov chain that performs a random walk in $\mathfrak{G}$ starting from the state G_0. Two states, G_i and G_j, are said to be adjacent if there is an alternating rectangle or hexagon in their adjacency matrices that after a switch would convert the first into the second. The transition probability matrix P consists of $P_{i,j} = \frac{1}{A_i}$ if G_i and G_j are adjacent, where A_i is the number of alternating rectangles and hexagons in the matrix of G_i, and $P_{i,j} = 0$, otherwise. From the above, it follows trivially that if G_j is reachable from G_i, G_i is also reachable from G_j, but a question rises: Is every state in $\mathfrak{G}$ reachable from every other? In other words, we have to prove that this Markov chain is irreducible. If so, we have to analyze it for periodicity. If the chain is aperiodic it would imply that with $\tau \to \infty$ the states G_τ are sampled from a unique distribution (termed *stationary* distribution) [7]. Thus one has to ask further: What is the stationary distribution and is it really the uniform one? Finally, if the answer to the first question is yes and we find the stationary distribution to be correct for our purposes, one will be interested in the convergence rate of the chain or what is the minimum value of τ, that would produce good sampling. All this is crucial for the correctness and the performance of the algorithm and will be the subject of the next couple of paragraphs.

The generation of G_0 is also interesting from algorithmic point of view. If the sequence D is not graphical it will be impossible to produce a single directed graph, so one has to check for it first. In practice D is derived from the real data, so one could use with the given graph itself as a starting point. Alternatively, if D is graphical one could generate one directed graph using a constructive algorithm based on the Gale-Ryser Theorem [12].

We will start with the proof of the claim that the so defined Markov chain is irreducible [20].

Theorem 2.1.1. *The space $\mathfrak{G}$ is connected upon switching along alternating rectangles and hexagons and the shortest path between any two directed graphs G_A and G_B with adjacency matrices A and B respectively is no more than $d(A, B)/2$.*

Proof. Let G_A and G_B be two arbitrary simple directed graphs from $\mathfrak{G}$. If $d(A, B) = 0$, the statement is true. Otherwise, let $d(A, B) > 0$. The further proof is by contradiction, therefore let us assume that there is no alternating rectangle or hexagon in either A or B so that a switch along it would shorten the Hamming distance between them.

Let $\mathcal{V_A}$ be the set of vertices of G_A and $\mathcal{V_A} = \{a_1, a_2, \ldots, a_n\}$, analogously $\mathcal{V_B} = \{b_1, b_2, \ldots, b_n\}$ and $in_{a_i} = in_{b_i} = in_i$ and $out_{a_i} = out_{b_i} = out_i, 1 \leq i \leq n$. We will form the following directed bipartite graph: $G = (\mathcal{V_G}, \mathcal{E_G})$ with $\mathcal{V_G} = \mathcal{V_A} \bigcup \mathcal{V_B}$ and $(a_i, b_j) \in \mathcal{E_G}$ if and only if $a_{i,j} = 1$ and $b_{i,j} = 0$ and $(b_i, a_j) \in E_G$ if and only if $a_{i,j} = 0$ and $b_{i,j} = 1$. Thus the edges of G represent the cells that differ in A and B and their number is equal to $d(A, B)$. Also, $\sum_{j=1}^{n} a_{i,j} = \sum_{j=1}^{n} b_{i,j}$ implies that the number of incoming edges equals the number of outgoing edges at each node a_i. Conversely, $\sum_{i=1}^{n} a_{i,j} = \sum_{i=1}^{n} b_{i,j}$ implies that the number of incoming edges equals the number of outgoing edges at each node b_j. Since the main diagonals of A and B contain structural zeros, it follows that $(a_i, b_i) \notin \mathcal{E_G}$ and $(b_i, a_i) \notin \mathcal{E_G}$ for all i. We will call the pairs (a_i, b_i) and (b_i, a_i) *diagonal.* Furthermore, we will call a path in G *proper* if all the vertices in it belong to different diagonal pairs. Our objective will be to prove that under these assumptions there could be no proper path with more than 2 edges.

Suppose that there exists at least one such proper path and without loss of generality let $\mu = a_{i_1}, b_{i_2}, a_{i_3}, b_{i_4}, \ldots$ be the longest one. The *length* of μ is defined as the number of edges in it. We will consider two cases: μ has odd or even length. If the length is odd, $\mu = a_{i_1}, b_{i_2}, a_{i_3}, b_{i_4}, \ldots, b_{i_{2k}}$ and $k \geq 2$ (the red arrows in Figure 2.2a). By their definition, the first three edges of the path correspond to the following statement: there are edges (i_1, i_2) and (i_3, i_4) in A but not in B and there is an edge (i_3, i_2) in B but not in A. Now there are four options for the edge (i_1, i_4): it is present in

B but not in A (Figure 2.2b), or it is neither in A nor in B (Figure 2.2c), or it is present in both A and B (Figure 2.2d), or it is present in A but not in B (Figure 2.2e). In the first case one can perform a switch along the alternating rectangle $\{i_1, i_3, i_2, i_4\}$ in either A or B that will shorten the Hamming distance between them by 4. In the second, the same nodes form an alternating rectangle only in A, so a switch along it would reduce the distance by 2. Case three is analogous, but the switch is performed in B and again the distance is reduced by 2. Therefore, the only case which is not contradictory to the assumption that there is no alternating rectangle in either A or B that could shorten the distance between them is the last one (Figure 2.2e). This means that there should be an edge (a_{i_1}, b_{i_4}) in G (the dashed arrow in Figure 2.2a). But now, by the same logic, it follows that there should be also an edge (a_{i_1}, b_{i_6}), because otherwise $\{i_1, i_3, i_2, i_6\}$ would form an alternating rectangle in A or B, and so recursively there should be k outgoing edges from a_{i_1} to each of the nodes b_{i_j} where $j = 2, 4, \ldots, 2k$. Since the number of incoming and outgoing edges are equal for each node, this means that there should be also k incoming edges at a_{i_1} and none of them can point from one of the nodes b_{i_j}, because there is already an edge between them that goes in the opposite direction. There will be at least one edge that points from a node b_{i_0} to a_{i_1} where i_0 is different from $i_1, i_3, \ldots, i_{2k-1}$. This is because there are k "a"-nodes in μ, but there could not be an edge between a_{i_1} and b_{i_1}, so at most $k-1$ "b"-nodes with the same indices can be a source of an edge pointing to a_{i_1}. Now the path $b_{i_0}, a_{i_1}, b_{i_2} \ldots, b_{i_{2k}}$ is proper and turns to be longer than μ, which is a contradiction to the maximality of μ.

So, let μ be of even length and $\mu = a_{i_1}, b_{i_2}, a_{i_3}, b_{i_4}, \ldots, b_{i_{2k}}, a_{i_{2k+1}}$, $k \geq 2$ (Figure 2.3). Again, it follows that there exist k outgoing edges at node a_{i_1} that point to $b_{i_2}, \ldots, b_{i_{2k}}$ and therefore should be at least k incoming edges at this node. Now, if one of the incoming edges at a_{i_1} has a source node that is different from $b_{i_3}, b_{i_5}, \ldots, b_{i_{2k+1}}$, it would mean that the path that starts from it and continues as μ would be proper and longer than μ. Since there could not be more than one edges between any two nodes in G there will be surely an edge from b_{i_3} to a_{i_1}. Now, let us look at μ from the other end. It follows in a similar manner that there should be k incoming edges at node $a_{i_{2k+1}}$, because each node b_{i_j} must point to $a_{i_{j+1}}, a_{i_{j+3}}, \ldots, a_{i_{2k+1}}$, otherwise it would be possible to switch along some alternating rectangle and reduce the distance between A and B. This implies that there should be also k outgoing edges at node $a_{i_{2k+1}}$. Again, they cannot point to one of the nodes $b_{i_2}, \ldots, b_{i_{2k}}$. In fact, if one of these edges points to a node different from $b_{i_1}, b_{i_3}, \ldots, b_{i_{2k-1}}$, it would create a proper path in G that is longer than μ. So, it turns out that there is an edge from $a_{i_{2k+1}}$ to b_{i_3}. But this means that

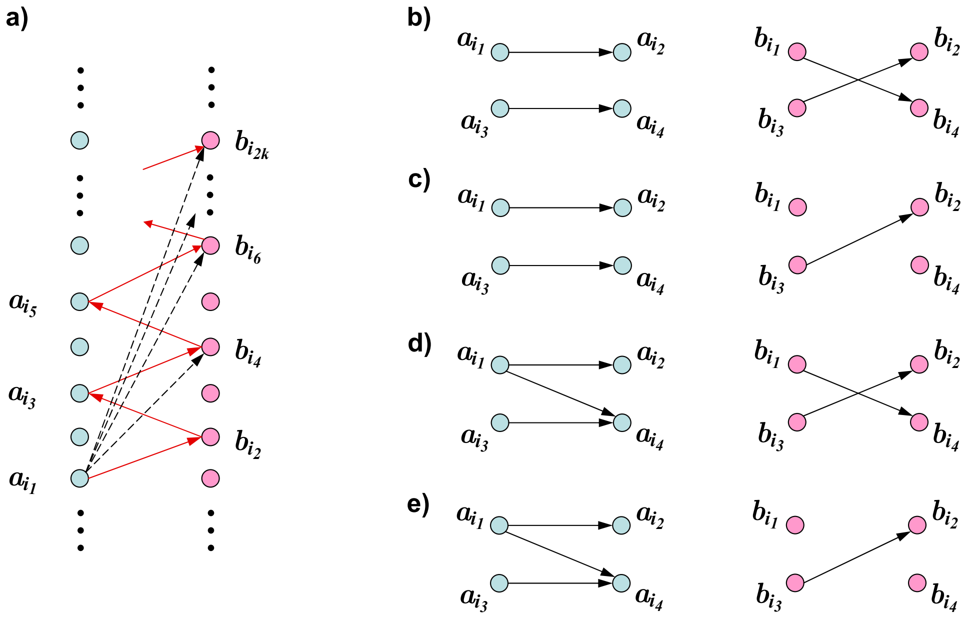

Fig. 2.2: a) The bipartite directed graph G and the path μ in the case that it consists of odd number of edges. The vertices from V_A are drawn cyan on the left side and from V_B - respectively magenta on the right. Vertices a_i and b_j are drawn on the same level if $i = j$. If $i \neq j$, the order of depicting the vertices does not matter in any way. The edges from μ are drawn in red. The existence of the dashed edges follows from the maximality of μ. b), c), d), and e) correspond to different possibilities in the digraphs A and B. On each picture A is displayed on the left side with cyan vertices and B - on the right with magenta.

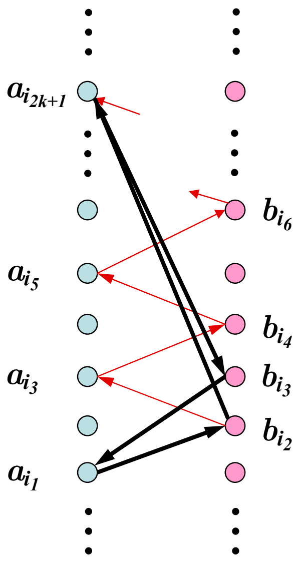

Fig. 2.3: The proper path μ in the case that it is of even length. The edges that belong to μ are drawn in red with exception of (a_{i_1}, b_{i_2}) which is a part of it as well. The other thick arrows represent edges that should exist following from the maximality of μ.

there is a closed cycle going through nodes $a_{i_1}, b_{i_2}, a_{i_{2k+1}}$ and b_{i_3} (the thick line in Figure 2.3), which is equivalent to saying that these nodes form an alternating rectangle in both A and B. A switch along it would reduce the distance between them by 4. Thus, the longest proper path in G under the assumption that it is impossible to convert A into B or vice versa with a series of switches consists of two edges at most.

Finally, there is one last thing to show to complete the proof. From the above it follows that, if there are edges (a_{i_1}, b_{i_2}) and (b_{i_2}, a_{i_3}), the only possibility to continue this path is with $(a_{i_3}, b_{i_1}), (b_{i_1}, a_{i_2}), (a_{i_2}, b_{i_3})$ and (b_{i_3}, a_{i_1}) (Figure 2.4a), because otherwise there would be a proper path that is longer than two. This corresponds to an alternating hexagon in both A and B (Figure 2.4b) and a switch along it would reduce the distance between them by 6. This obviously contradicts to the assumption that it cannot be shortened by a switch along some alternating hexagon or rectangle and therefore proves that at least one such switch exists. Each switch reduces the distance by at least 2, so one needs at most $d(A, B)/2$ steps to transform A into B and vice versa. □

Before proceeding with the next question about the Markov chain, let us analyze a little bit what we just proved. Every proper path in G that consists

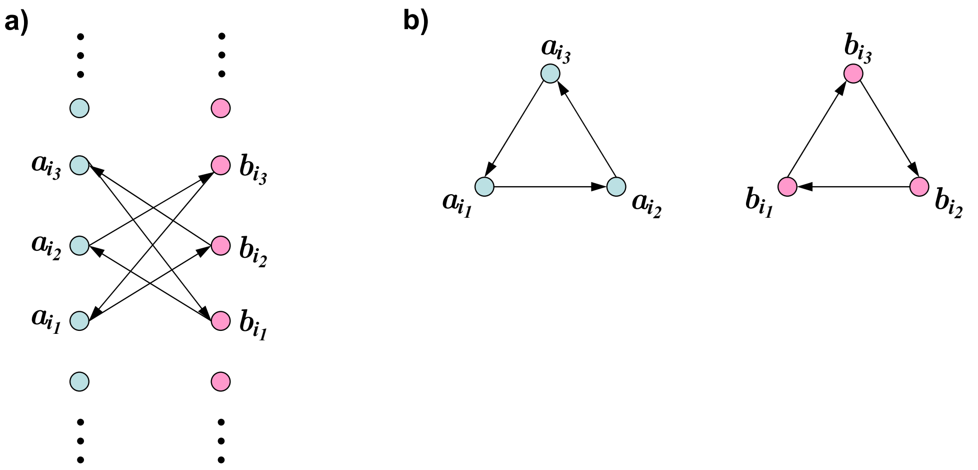

Fig. 2.4: a) The only possible path in G under the assumption that there is no alternating rectangle in either A or B that can shorten the distance between them. b) The corresponding edges in A and B.

of more than two edges can be shortened by a switch along an alternating rectangle. The case of Figure 2.4 turns to be the only one that needs a switch along an alternating hexagon. Thus, at first glance it seems that if the graphs A and B contain a cycle going through nodes i_1, i_2 and i_3 but in different directions, one would need to switch along the alternating hexagon $\{i_1, i_2, i_3\}$ in order to convert A into B. This is true, but if there is at least one more edge in A and B respectively which is not between i_1, i_2 and i_3, one can reverse the direction of the cycle only with switches along alternating rectangles. Figure 2.5 shows an example of how this could be done. Notice that the first step of this process transforms A into A_1 with $d(A_1, B) = d(A, B)$, so the Hamming distance is not reduced. The subsequent ones reduce it by 2 and 4 then. Thus,

Corollary 2.1.1. *The only case where the space $\mathfrak{G}$ is not connected upon switches along alternating rectangles alone is if $in_{i_1} = in_{i_2} = in_{i_3} = 1$ and $out_{i_1} = out_{i_2} = out_{i_3} = 1$ for three distinct integers $i_1, i_2, i_3 \in [1, n]$ and $in_j = out_j = 0$ for all integer j, such that $j \in [1, n] \setminus \{i_1, i_2, i_3\}$.*

In this case $\mathfrak{G}$ consists of two graphs. The first one has a single three node cycle and all other nodes do not participate in any edges. The second one is isomorph to the first one but the cycle goes in the opposite direction. Fortunately, this is rarely the case for real life networks.

Furthermore, another corollary from Theorem 2.1.1 is

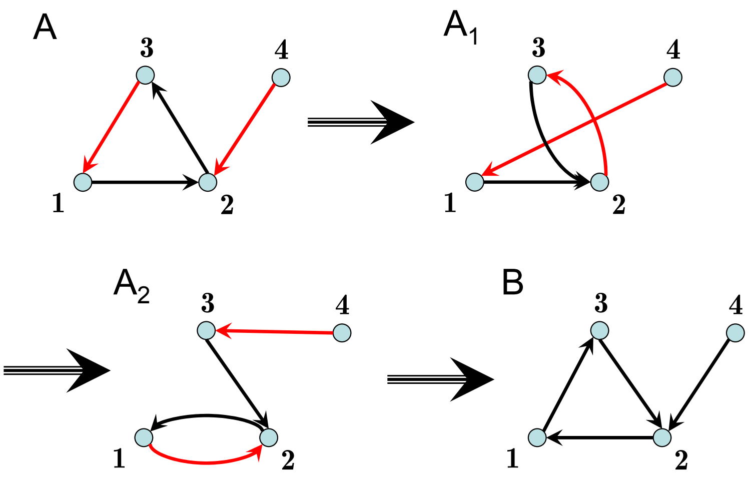

Fig. 2.5: Example of reversing the direction of a three node cycle by using an auxiliary edge. The edges that are swapped at each step are drawn in red.

Corollary 2.1.2. *The space of all directed graphs with loops with a given degree sequence is connected upon switches along alternating rectangles alone.*

An easy way to prove this is given in [20]. It was also shown that the spaces of undirected labeled and unlabeled simple graphs, connected graphs, multigraphs and pseudographs with finite and infinite degree sequences are connected upon *switching* defined as interchanging the ends of two edges that produces a representative from the same space [25].

Now we shall analyze the Markov chain for periodicity. In this form it could be periodic or aperiodic depending on the degree sequence. It is easy to see that if the chain has a period it should be 2. This is true, because after going from state G_i to G_j, it is always possible to go back with a switch along the same alternating rectangle or hexagon, respectively. Figure 2.6 shows examples of both. In Figure 2.6b the chain can reach every state in an odd number of switches from an arbitrary initial state and therefore it is aperiodic. Instead of delving deeper into this question we will simply modify the chain in a way that it is always aperiodic by letting $P_{i,i} \neq 0$ for at least one i. This can be done in different ways (see for example [15, 20]). One possibility that deserves attention (because it leads to biased sampling) is to let $P_{i,j} = \frac{1}{A_i+1}$ for $i = j$ and if G_i and G_j are adjacent, and $P_{i,j} = 0$, otherwise. This means that the chain stays in its current state or moves to an adjacent state with the same probability. Since it is irreducible and aperiodic it follows that the distribution of the states tends to the stationary

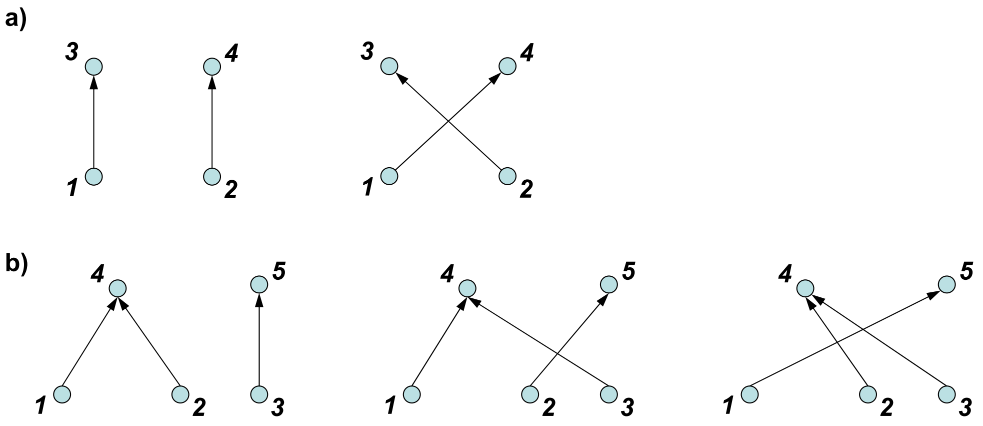

Fig. 2.6: Realizations of a degree sequence, where the Markov chain from Algorithm 2.1.1 is a) periodic or b) aperiodic.

distribution with $\tau \to \infty$ [7]. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$ be the vector of the stationary distribution. To find it we need to solve $\pi P = \pi$ for π_i:

$$\pi_j = \sum_{i=1}^{N} \pi_i P_{i,j}.$$

Since the number of non-zero elements in column j equals the number of non-zero elements in row j in P, the unique solution is

$$\pi_i = \frac{A_i + 1}{\sum\limits_{k=1}^{N} A_k}.$$

This, in general, is not equal to $\frac{1}{N}$, so we are in trouble.

Probably the best way to enforce the chain to be aperiodic stems from the implementation in [15]. The algorithm does not consider alternating hexagons, but as we saw this is a problem only in few pathological cases. It picks two edges at random and performs the switch if it does not cause a self- or a multiple edge to be created. If that happens, it tries with another pair. In this way

$$P_{i,j} = \begin{cases} \binom{E}{2}^{-1} = \frac{2}{E(E-1)}, & \text{if i and j are adjacent,} \\ F_i \frac{2}{E(E-1)}, & \text{if i = j,} \\ 0, & \text{otherwise,} \end{cases}$$

where F_i is the number of ways to create a self- or a multiple edge at state i plus the number of switches that do not change the graph. Now, for our purpose it will be sufficient that F_i is not 0 for at least one i. Unfortunately, there are some rare cases where this is not so. Then, the graph should consist of a layer of source nodes and a layer of target nodes. If there is at least one node j with $in_j \neq 0$ and $out_j \neq 0$ it would be possible to create a self-edge at it and so $F_i \neq 0$ for all i. Furthermore, each source node is allowed to have only one outgoing edge and each target node one incoming edge. Otherwise, if we switch two edges with the same source or target vertex, the state of the Markov chain will remain unchanged. This condition assures also that no multiple edges can be created. One has to say that this kind of degree sequence is also not typical for real networks. Nevertheless, for the sake of completeness one could let

$$P_{i,j} = \begin{cases} \frac{2}{E(E-1)+1}, & \text{if i and j are adjacent,} \\ \frac{2F_i+1}{E(E-1)+1}, & \text{if i = j,} \\ 0, & \text{otherwise,} \end{cases}$$

and thus overcome this problem. In both cases the transition probability matrix is doubly stochastic, because the column sum is also equal to 1. Therefore it follows that the stationary distribution is the uniform one, so $\pi_i = \frac{1}{N}$. The proof of this statement in general could be found in [7].

Finally, we are ready to formulate the second version of the switching algorithm, which generates uniformly directed graphs in all but a few cases that are uninteresting in the context of network motif searching.

Algorithm 2.1.2.

1. *Choose an integer $\tau > 0$.*

2. *Generate one directed graph (denoted as G_0) that has the given degree sequence.*

3. *Pick two edges from G_0 and swap their ends if this does not result in a self- or a multiple edge. Otherwise, do nothing. In both cases denote the digraph that results as G_1.*

4. *Do step 3. recursively τ times.*

5. *Output the digraph G_τ.*

The last question we want to answer is how large should τ be in order to be sure that the stationary distribution is close enough to the uniform. This highly depends on the degree sequence D. Intuitively, if N or F_i are large, one would need more switches. Starting from G_0, the probability that a digraph G_k is output at step τ is $P^{\tau}_{0,k}$, where P^{τ} is the τ^{th} power of P. We would like to find a value for τ so that the difference $\epsilon = |\frac{1}{N} - P^{\tau}_{0,k}|$ is small enough. The problem is that it seems quite difficult to calculate the values of $P^{\tau}_{0,k}$, because they depend on the cardinality of $\mathfrak{G}$, N, and on the values F_i. There is a series of papers that discuss the issue of finding the number of all possible simple directed graphs with given degree sequence [1, 2, 11, 23], but the best known result is an approximation of N for $n \to \infty$. To the best of our knowledge, there is no study of the values F_i and probably their analysis is also complicated. Therefore, in practice one sets a fixed number of switches to be performed. Some simulations [15] show that $\tau = 100 * E$ is sufficient that the stationary distribution is close enough to the uniform one.

2.2 The Matching Algorithm

The second algorithm, termed "the matching algorithm", implements a different strategy. It starts with an empty digraph and generates randomly edges until a digraph is formed that has the desired degree sequence. Like the previous one it has several implementations, not all of which sample uniformly. The matching algorithm uses the concept of stubs, that represent edges where either the source or the target vertex is unknown. We will start with the first version of the algorithm:

Algorithm 2.2.1.

1. *Create an empty digraph $G = (\mathcal{V}, \mathcal{E})$ with n nodes. To each node assign a proper number of incoming and outgoing stubs according to the desired degree sequence.*

2. *If there are no free stubs, go to step 4. Otherwise, from the sets of free outgoing and incoming stubs pick one from each at random. Let these stubs belong to nodes $\mathbf{i}$ and $\mathbf{j}$ respectively.*

3. *If $\mathbf{i} \neq \mathbf{j}$ and $(\mathbf{i},\mathbf{j}) \notin \mathcal{E}$, let $\mathcal{E} = (\mathbf{i},\mathbf{j}) \bigcup \mathcal{E}$ and go to step 2. Otherwise, go to step 1.*

4. *Output G.*

We will refer to this algorithm as the *unmodified version of the matching algorithm.* It is easy to see that it can generate every possible directed graph with the given sequence. Furthermore, we will prove that it also samples uniformly from $\mathfrak{G}$.

Theorem 2.2.1. *Let $D = \langle (in_1, out_1), (in_2, out_2), \ldots, (in_n, out_n) \rangle$ be a given degree sequence. All simple directed graphs with degree sequence D have the same probability to be generated by the unmodified version of the matching algorithm.*

Proof. Let $G = (\mathcal{V}, \mathcal{E})$ be an arbitrary digraph from $\mathfrak{G}$. Let us assume for now that it was generated by the algorithm without going to step 1 from step 3. This means that no attempt to create a self- or multiple edge was made. The overall probability to generate G is then the sum of the probabilities to create its edges over all possible ways to do so. Suppose the algorithm picked the edges in some order. We will label them with e_i, $i = 1 \ldots E$ where i is the time point at which the edge e_i was created. Now we will compute the probability $P(e_1, e_2, \ldots, e_E)$ to pick the edges $e_1, e_2, \ldots, e_E$ in exactly this order:

$$P(e_1, e_2, \ldots, e_E) = P(e_1)P(e_2|e_1)P(e_3|e_1, e_2) \ldots P(e_E|e_1, e_2, \ldots, e_{E-1}).$$

Let $in_{\delta,i}$ and $out_{\delta,i}$ be the number of free incoming and outgoing stubs at node δ before the i^{th} random choice. Then the probability to create the edge e_i pointing from node e_i^{out} to node e_i^{in} at the i^{th} random choice is

$$P(e_i|e_1, e_2, \ldots, e_{i-1}) = \frac{out_{e_i^{out},i}}{\sum\limits_{\delta=1}^{n} out_{\delta,i}} \, \frac{in_{e_i^{in},i}}{\sum\limits_{\delta=1}^{n} in_{\delta,i}} = \frac{out_{e_i^{out},i}\, in_{e_i^{in},i}}{(E-i+1)^2}.$$

Therefore,

$$\begin{aligned} P(e_1, e_2, \ldots, e_E) &= \frac{(out_{e_1^{out},1}\, in_{e_1^{in},1})}{E^2} \, \frac{(out_{e_2^{out},2}\, in_{e_2^{in},2})}{(E-1)^2} \ldots \frac{(out_{e_E^{out},E}\, in_{e_E^{in},E})}{1} = \\ &= \frac{\prod\limits_{i=1}^{E} (out_{e_i^{out},i}\, in_{e_i^{in},i})}{(E!)^2}. \end{aligned} \tag{2.1}$$

Let us take a node δ from G that has out_δ outgoing edges. This means that $e_i^{out} = \delta$ in exactly out_δ of the terms in Equation 2.1. Let $\tau_{\delta,i}$ be the i^{th} point of time where an outgoing stub from δ is chosen. Then,

$$\begin{aligned} & out_{\delta,\tau_{\delta,1}} = out_{\delta,1}, \\ & out_{\delta,\tau_{\delta,2}} = out_{\delta,1} - 1, \\ & out_{\delta,\tau_{\delta,3}} = out_{\delta,1} - 2, \\ & , \ldots, \\ & out_{\delta,\tau_{\delta,i}} = out_{\delta,1} - i + 1, \\ & , \ldots, \\ & out_{\delta,\tau_{\delta,out_\delta}} = 1. \end{aligned}$$

As we see, the values of $out_{\delta,\tau_{\delta,i}}$ do not depend on $\tau_{\delta,i}$. They will just decrease by 1 every time we pick an outgoing stub from δ. Since multiplication is commutative, we can group them together and rewrite Equation 2.1 in the following form:

$$P(e_1, e_2, \ldots, e_E) = \frac{\prod_{i=1}^{E} (out_{e_i^{out},i}\, in_{e_i^{in},i})}{(E!)^2} =$$

$$= \frac{\prod_{\delta \in Out(G)} (out_\delta(out_\delta - 1)\ldots 2.1) \prod_{i=1}^{E} in_{e_i^{in},i}}{(E!)^2} = \frac{\prod_{\delta=1}^{n} out_\delta! \prod_{i=1}^{E} in_{e_i^{in},i}}{(E!)^2},$$

where $Out(G)$ is the set of nodes in G that have at least one outgoing edge. Applying the same logic we can group together the values of $in_{e_i^{in},i}$, so Equation 2.1 becomes

$$P(e_1, e_2, \ldots, e_E) = \frac{\prod_{\delta=1}^{n} out_\delta! \prod_{\delta=1}^{n} in_\delta!}{(E!)^2}, \tag{2.2}$$

Now, if the algorithm picks the edges $e_1, e_2, \ldots, e_E$ in a different order, we can always rewrite the probability to do so in the form of Equation 2.4. Therefore, the probability to pick the edges in any of these orders is equal to $P(e_1, e_2, \ldots, e_E)$ in it. Since there are $E!$ possible different orders, the overall probability that the digraph G is generated is

$$P(G) = E!\, P(e_1, e_2, \ldots, e_E) = E!\, \frac{\prod_{\delta=1}^{n} out_\delta!\, in_\delta!}{(E!)^2} = \frac{\prod_{\delta=1}^{n} out_\delta!\, in_\delta!}{E!}. \tag{2.3}$$

Since G was chosen arbitrary and $P(G)$ depends only on the degree sequence D, all digraphs from $\mathfrak{G}$ have the same probability to be generated if the algorithm is at step 1 and comes to completion without returning to it. Now, if that happens, it discards the partial networks and starts everything from scratch. This action does not give credit to any of the members of $\mathfrak{G}$, because it brings the algorithm to a state where all digraphs are again equally probable if it comes to completion without returning to step 1. Therefore, if the algorithm terminates the output will be a uniformly chosen digraph G from $\mathfrak{G}$. □

Here we will make a brief excursion from the further analysis of the unmodified version of the matching algorithm. Suppose, we change the algorithm in a way that it does not check if a self- or a multiple edge is created at step 3 and just proceeds with the generation process until a representative from the set of directed pseudographs is created. In pseudocode this can be expressed as:

Algorithm 2.2.2.

```
1. Create an empty directed pseudograph $\Psi = (\mathcal{V}, \mathcal{E})$ with $n$ nodes. To each node assign a proper number of incoming and outgoing stubs according to the desired degree sequence.
2. If there are no free stubs, go to step 4. Otherwise, from the sets of free outgoing and incoming stubs pick one from each at random. Let these stubs belong to nodes $\mathbf{i}$ and $\mathbf{j}$ respectively.
3. Let $\mathcal{E} = (\mathbf{i}, \mathbf{j}) \bigcup \mathcal{E}$ and go to step 2.
4. Output $\Psi$.
```

Will Ψ be uniformly chosen? Surprisingly, no. In general, the pseudographs that contain multiple edges will be sampled with lower probability than $P(G)$ from Equation 2.3. For example, in Figure 2.7 the cases f) and g) are sampled with probability $\frac{1}{12}$ and the rest with $\frac{1}{6}$. Why is it so? Let us see how case f) can be generated. We will refer to the two edges between node 1 and 4 as $(1,4)^{\mathbf{1}}$ and $(1,4)^{\mathbf{2}}$ in order to distinguish between them. Let us assume that case f) was generated in the following way: first, the edge $(1,4)^{\mathbf{1}}$, then $(1,4)^{\mathbf{2}}$ and $(2,3)$, and finally $(3,2)$. The probability to connect the two stubs that are the source and the target of $(1,4)^{\mathbf{1}}$ is $\frac{1}{4}.\frac{1}{4}$. The probability to connect the stubs of the second edge is respectively $\frac{1}{3}.\frac{1}{3}$ and so on. So, the probability to create f) connecting exactly these stubs is

$$\frac{1}{(4!)^2}.$$

Since we can pick these edges in 4! ways and since we can connect the two outgoing stubs of node 1 with the two incoming stubs of node 4 in two ways, the overall probability to create f) is

$$4!.2.\frac{1}{(4!)^2} = \frac{1}{12}.$$

The probability of case g) can be calculated analogously and is also $\frac{1}{12}$. Generally, as in the case of simple directed graphs, the overall probability to generate a pseudograph Ψ is equal to the probability to create its edges in

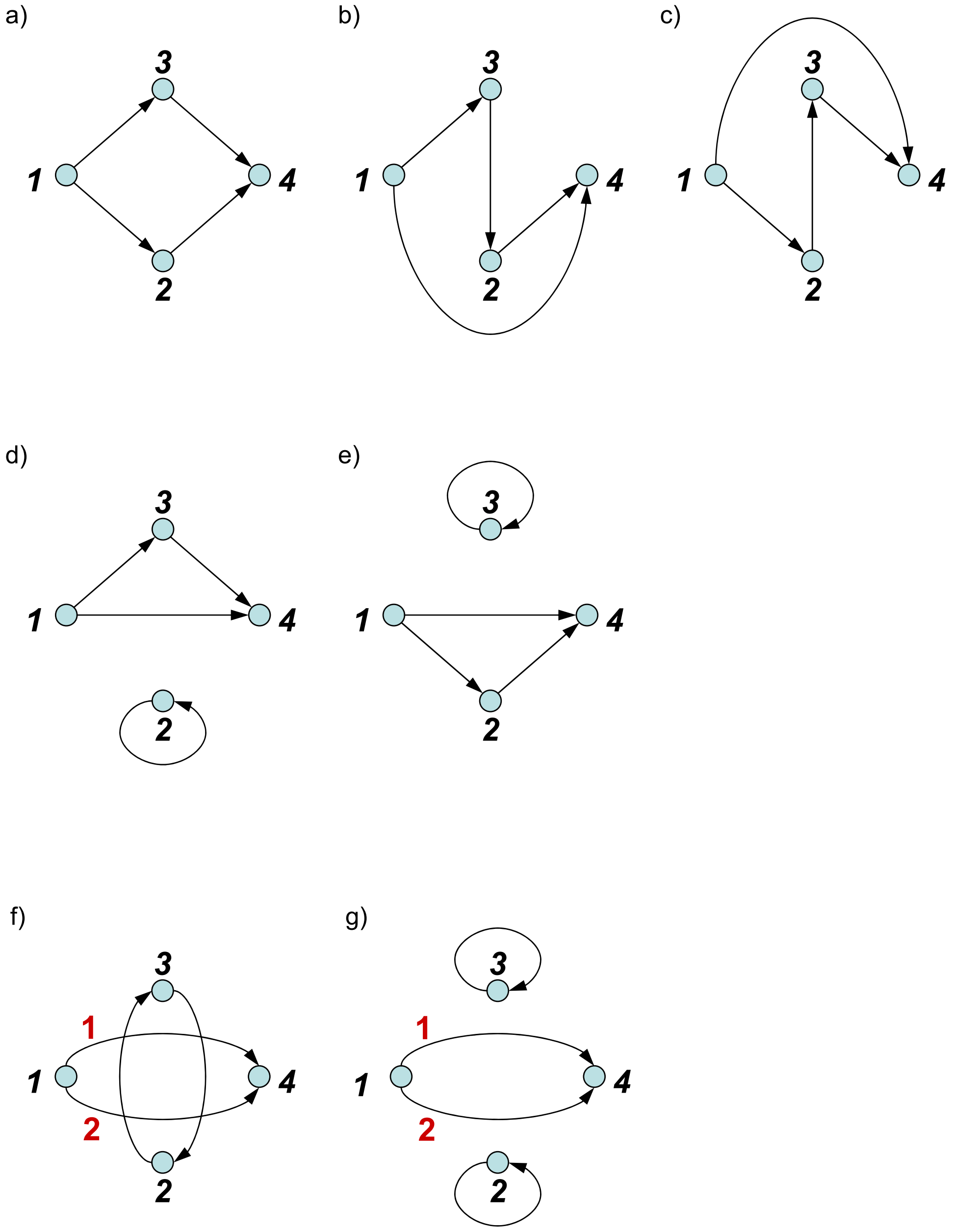

Fig. 2.7: All realizations of the degree sequence $\langle(0,2),(1,1),(1,1)(2,0)\rangle$ in the space of directed pseudographs. In f) and g) the two edges between node 1 and 4 are labeled 1 and 2 respectively in order to distinguish between them. These two cases are undersampled by Algorithm 2.2.2.

one particular order, $\frac{1}{(E!)^2}$, multiplied by the number of all possible orders to pick them. There are $E!$ ways to pick the edges that connect the same pairs of stubs and some number of ways to choose pairs of stubs that produce the same edges. Let us see how to calculate the latter. If we have two nodes i and j with out_i outgoing and in_j incoming stubs respectively and there are k multiple edges between them, there are

$$\binom{out_i}{k}\binom{in_j}{k}k!$$

ways to connect the stubs and produce them. Let the pseudographs be represented by an adjacency matrix M, where $m_{i,j}$ equals the number of edges pointing from node i to node j. Then, there are

$$\prod_{j=1}^{n}\binom{out_i - \sum_{l=1}^{j-1} m_{i,l}}{m_{i,j}}$$

ways to partition the outgoing stubs of node i to create its outgoing edges. Conversely, there are

$$\prod_{j=1}^{n}\binom{in_i - \sum_{l=1}^{j-1} m_{l,i}}{m_{j,i}}$$

ways to partition the incoming stubs at node i to create its incoming edges. Like in Equation 2.3 we can group together these terms and thus obtain the overall probability to generate a pseudograph Ψ with Algorithm 2.2.2:

$$P(\Psi) = \frac{\prod_{\delta=1}^{n}\prod_{j=1}^{n}\binom{out_\delta - \sum_{l=1}^{j-1} m_{\delta,l}}{m_{\delta,j}}\prod_{j=1}^{n}\binom{in_\delta - \sum_{l=1}^{j-1} m_{l,\delta}}{m_{j,\delta}}\prod_{j=1}^{n} m_{\delta,j}!}{E!}. \tag{2.4}$$

This probability varies depending on the amount of multiple edges in Ψ. In general, the more multiple edges are in Ψ the less is the probability to be generated. Nevertheless, if we let all non-zero cells of M be equal to 1 (which means that there are no multiple edges in Ψ) it reduces to Equation 2.3. Therefore, to create a sample with Algorithm 2.2.2 that is big enough and to take only the simple directed graphs from it will have the same effect as running Algorithm 2.2.1.

After we proved the correctness of the unmodified version of the matching algorithm it is time to analyze its performance. As expected, it has one major drawback: using degree sequences from real networks one has to wait "forever" to create an ensemble of randomized digraphs, because multiple and self-edges are created with high probability. The question about the time needed is equivalent to the one of how many times does one need to run Algorithm 2.2.2 until a simple directed graph is generated. In this context Algorithm 2.2.2 represents a Bernoulli trial with probability P_D to output a simple directed graph and $1 - P_D$ to output a directed graph that has loops or multiple edges. The number of runs, k, follows the geometric distribution with expected value $\frac{1-P_D}{P_D}$. P_D is equal to the number of possible simple directed graphs with the given degree sequence, $N = |\mathfrak{G}|$, multiplied by $P(G)$ from Equation 2.3. The big difficulty is to calculate the exact value of N. There is a series of papers that discuss this issue in detail [1, 2, 11, 23]. Theoretically, N could range from 0 to $\frac{1}{P(G)}$ depending on the degree sequence. To analyze the average case may become extremely difficult and probably not very informative for the practice. Instead, we will focus on the performance given degree sequences that are close to the ones of some real networks. If the in- and outdegrees are bounded by a constant, it is possible to show that

$$N \approx \frac{E!}{\prod\limits_{i=1}^{n} in_i! out_i!} exp\left(-\frac{\left(\sum\limits_{i=1}^{n} in_i^2 - E\right)\left(\sum\limits_{i=1}^{n} out_i^2 - E\right)}{2E^2}\right)$$

for $n \rightarrow \infty$ [1, 2]. Let $in_i, out_i \leq K$ for a constant K and $1 \leq i \leq n$. Then, asymptotically

$$\begin{aligned} P_D &\approx exp\left(-\frac{\left(\sum\limits_{i=1}^{n} in_i^2 - E\right)\left(\sum\limits_{i=1}^{n} out_i^2 - E\right)}{2E^2}\right) \geq \\ &\geq exp\left(-\frac{\left(\sum\limits_{i=1}^{n} K^2 - nK\right)\left(\sum\limits_{i=1}^{n} K^2 - nK\right)}{2(nK)^2}\right) = \\ &= exp\left(-\frac{(K-1)^2}{2}\right). \end{aligned}$$

This approximation shows that P_D tends to a constant if one increases the number of nodes but keeps the maximum in- and outdegrees fixed. Nevertheless, this constant decreases faster than exponentially if one increases their upper bound. So, the expected number of runs will be

$$k = \frac{1 - P_D}{P_D} \approx exp \left(\frac{\left(\sum_{i=1}^{n} in_i^2 - E \right) \left(\sum_{i=1}^{n} out_i^2 - E \right)}{2E^2} \right) - 1 \leq$$
$$\leq exp \left(\frac{(K-1)^2}{2} \right) - 1. \tag{2.5}$$

Although not very rigorous, this analysis indicates why Algorithm 2.2.2 needs a lot of time to generate a simple directed graph in some cases (eg. regular graphs). The presence of hubs, which is typical for many real networks, increases more than exponentially the expected number of runs and makes respectively the unmodified version of the matching algorithm inefficient.

To compensate for this, the algorithm was changed [14, 15, 16, 22] in a way that, instead of discarding the partial network, it tries to find another pair of stubs that causes no conflicts after connection. If it fails after a predefined number of times, the whole process starts from beginning as in the unmodified version. The procedure can be stated as the following

Algorithm 2.2.3.

1. *Choose an integer $\varepsilon \geq 0$.*
2. *Create an empty digraph $G = (\mathcal{V}, \mathcal{E})$ with n nodes. To each node assign a proper number of incoming and outgoing stubs according to the desired degree sequence.*
3. *If there are no free stubs, go to step 5. Otherwise, from the sets of free outgoing and incoming stubs pick one from each at random. Let these stubs belong to nodes $\mathbf{i}$ and $\mathbf{j}$ respectively.*
4. *If $\mathbf{i} \neq \mathbf{j}$ and $(\mathbf{i}, \mathbf{j}) \notin \mathcal{E}$, let $\mathcal{E} = (\mathbf{i}, \mathbf{j}) \bigcup \mathcal{E}$ and go to step 3. Otherwise, decrement ε. If $\varepsilon < 0$, go to step 2. Else, go to step 3.*
5. *Output G.*

Although this solution looks intuitive, it conceals one insidious pitfall. The result is biased, because some combinations are favored over another. When the algorithm creates a self- or a multiple edge at some point, it does not start from the beginning by discarding the partial network. Instead, it undoes the mistake and increments a counter that stores the number of mistakes made up to the current point. Only if this number is more than some predefined threshold ε, it gives up and goes to step 2. Otherwise it returns to the previous state of the configuration and tries with the same probabilities to pick a pair of the available in- and outgoing stubs, thus giving the whole process another chance to come to completion. It does not memorize the bad choices, so it is possible to make the same or another mistake again and again. Even worse, it is still possible to come to a dead-end state where all pairs of in- and outgoing stubs represent self- or a multiple edges. In this case the edge that is the actual reason for the failure was placed somewhere in the past and retrying only wastes time. Thus, modifying the algorithm in this way does not guarantee that it will complete successfully. It just increases the probability of this to happen. The problem is that now it can generate the same digraph in more than $E!$ ways. More precisely, it is possible to create it with 0, 1, 2 and so on up to ε mistakes. In general, if $P_{x,i}$ is the probability to create the digraph x with i mistakes, where $0 \leq i \leq \varepsilon$, the overall probability to create x will be $\sum_{i=0}^{\varepsilon} P_{x,i}$. Depending on the degree sequence it may happen that these sums are not equal for all $x \in \mathfrak{G}$.

Here we will discuss an example that illustrates it. Suppose $\varepsilon = 1$, which means that during the whole process we allow only one mistake to be undone. Consider the following degree sequence: $\langle(0,2),(1,1),(1,1),(2,0)\rangle$ and all its realizations (Figure 2.7 a), b) and c)). We will now compute their overall probabilities P_a, P_b, and P_c, respectively. P_a is equal to the probability to create the digraph from Figure 2.7a) without mistakes, $P_{a,0}$, plus the probability to create it with one mistake, $P_{a,1}$. From Equation 2.3 it follows that

$$P_{a,0} = \frac{2!.2!.1!.1!}{4!} = \frac{1}{6}.$$

To calculate $P_{a,1}$ we have to sum over the probabilities of all possibilities (see Figure 2.8) to make a mistake at some step of the algorithm: $\sum_{i=1}^{4} P_{a,1,i}$, where $P_{a,1,i}$ is the probability to create the digraph from Figure 2.7a) with making one mistake at step i. These probabilities are

$$\begin{aligned}
P_{a,1,1} &= 2.\frac{1}{4}\frac{1}{4}.P_{a,0} = \frac{1}{8}P_{a,0} \\
P_{a,1,2} &= \frac{1}{3}\frac{1}{3}.P_{a,0} = \frac{1}{9}P_{a,0} \\
P_{a,1,3} &= 2.2.2.\frac{1}{2}\frac{1}{2}.\frac{P_{a,0}}{4!} = \frac{1}{12}P_{a,0} \\
P_{a,1,4} &= 0.
\end{aligned}$$

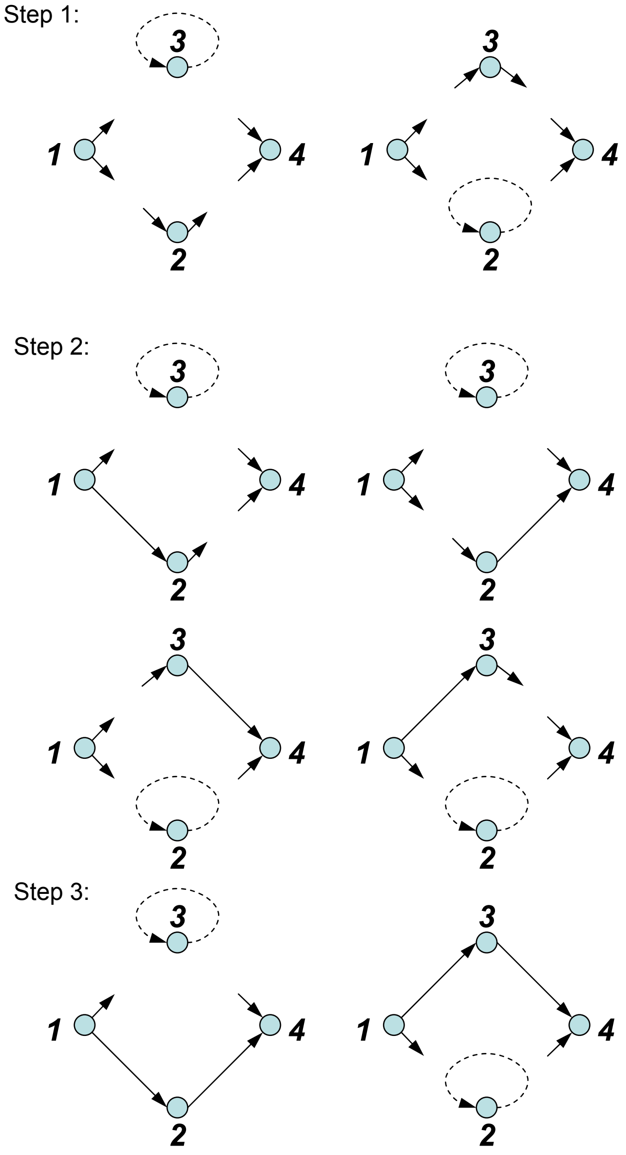

Fig. 2.8: All possibilities to create Fig. 2.7a) with one mistake at some step

So,

$$P_a = P_{a,0} + \frac{1}{8}P_{a,0} + \frac{1}{9}P_{a,0} + \frac{1}{12}P_{a,0} = \frac{95}{432}.$$

Now, let us calculate the probability to generate Figure 2.7b), P_b. Again, from Equation 2.3 we have that $P_{b,0} = P_{a,0} = \frac{1}{6}$. The possibilities to create this digraph with one mistake at some step are depicted in Figure 2.9. The probabilities $P_{b,1,i}$ are

$$\begin{aligned}
P_{b,1,1} &= 2.\frac{1}{4}\frac{1}{4}.P_{b,0} = \frac{1}{8}P_{b,0}\\
P_{b,1,2} &= (3\frac{1}{3}\frac{1}{3}).\frac{P_{b,0}}{4} + (2\frac{1}{3}\frac{1}{3}).\frac{P_{b,0}}{4} = \frac{5}{36}P_{b,0}\\
P_{b,1,3} &= 3.2!.2!(\frac{1}{2}\frac{1}{2}).\frac{P_{b,0}}{4!} = \frac{1}{8}P_{b,0}\\
P_{b,1,4} &= 0.
\end{aligned}$$

So,

$$P_b = P_{b,0} + \frac{1}{8}P_{b,0} + \frac{5}{36}P_{b,0} + \frac{1}{8}P_{b,0} = \frac{100}{432}.$$

Finally, because Figure 2.7b) and c) are isomorph, $P_c = P_b = \frac{100}{432}$.

This example shows that using Algorithm 2.2.3 changes the probabilities of the different possible digraphs with a given degree sequence in a way that they may be not equal anymore. It is still an open question how to estimate the bias for a given degree sequence and a threshold ε.

2.3 The "Go with the Winners" Algorithm

The third algorithm for generating random directed graphs is based on the "Go with the Winners" strategy introduced by David Aldous and Umesh Vazirani in [26]. Their original idea was illustrated on the problem of finding deep vertices on a tree. It could be described as follows: Given a tree and an integer d, find a vertex at depth d by traversing the tree from the root. Informally one could imagine a particle that is moved from node to node starting from the root without backtracking until it reaches the desired depth. At each step a successor is chosen in a random fashion.

The easiest way to solve the problem is just to let one particle move downwards. If its path stops at a leaf and consists of less than d nodes, discard the solution and start again from the root. To boost up the performance one could let more, say B, particles to move simultaneously and independently. If one of them reaches a vertex of depth d, the process stops and outputs the vertex. In their paper, Aldous and Vazirani refer to this as Algorithm 0.

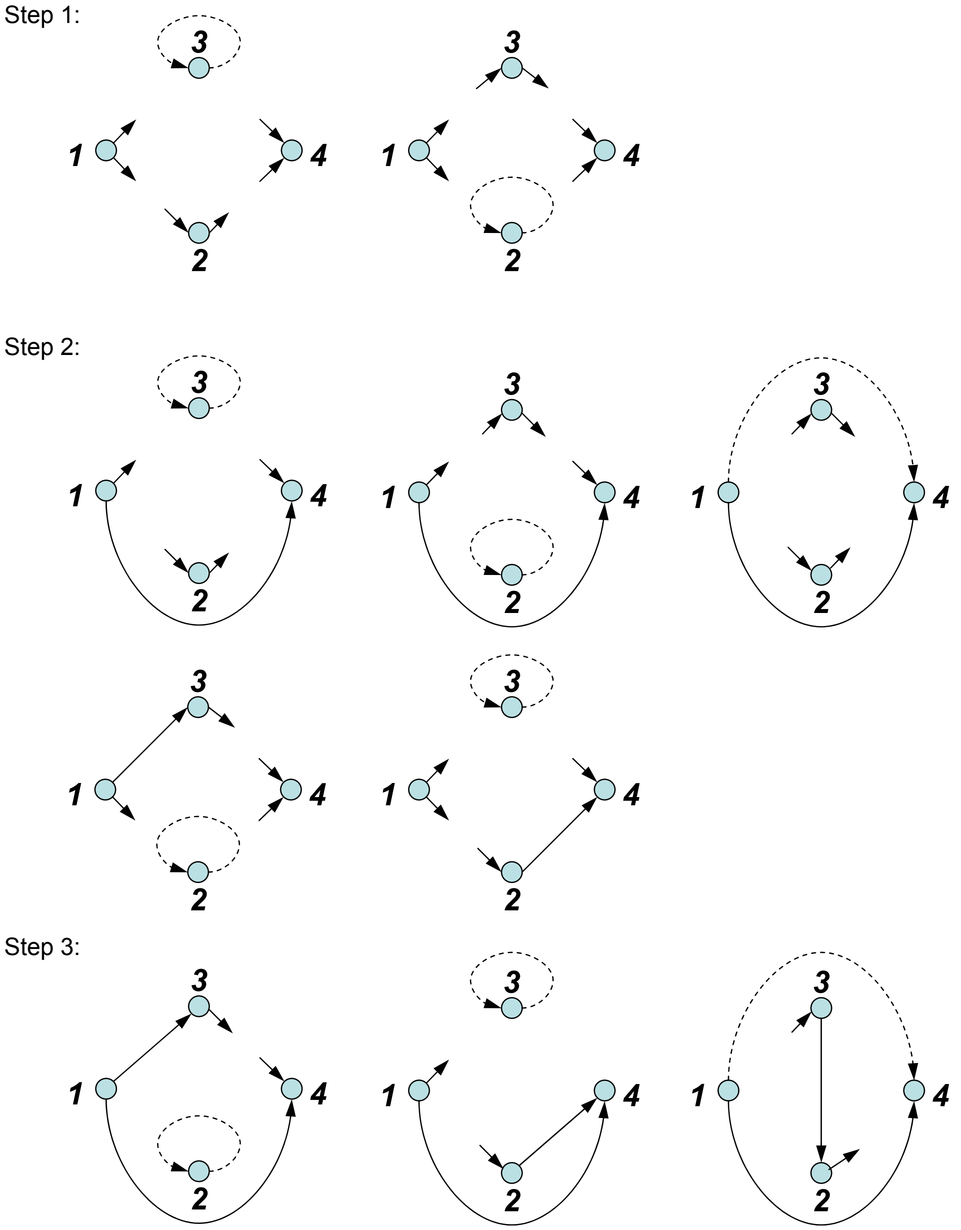

Fig. 2.9: All possibilities to create Fig. 2.7b) with one mistake at some step

Even by introducing this parallelism, Algorithm 0 is very inefficient in many topologies. Therefore, the authors suggest to allow interactions between the particles during the traversal process. For example, if there are $k_i < B$ particles stuck at leaf nodes at step i, distribute them randomly at the positions of the rest $B - k_i$ particles. This might be viewed also as discarding the stuck particles and cloning the rest in such a way that there are again B particles that can continue to traverse the tree. This trick is at the core of the ”Go with the Winners” strategy, to which implementation the authors refer as Algorithm 1. It incorporates a ”survival of the fittest” rule known from genetic algorithms and it is possible to show that the performance may increase significantly therewith.

”Go with the Winners” algorithms are known for many applications since then [15, 17, 18]. The one that is interesting for our purposes is the generation of random directed graphs [15]. To see how, let us take a look at the unmodified version of the matching algorithm (Algorithm 2.2.1). Clearly, the whole process of connecting incoming and outgoing stubs represents a decision tree, where the inner nodes stand for configurations with free stubs and the leaves are either failures due an attempt to create a self- or multiple edge or digraphs that have the desired degree sequence. Each branch corresponds to a choice of a pair of stubs that are going to be connected. The root has E^2 successors, since there are E^2 possible pairs of stubs at the beginning. The number of successors at each non-leaf node at depth i is $(E - i)^2$, but the proportion of leafs among them is hard to calculate because it depends explicitly on the given degree sequence. Also, if the space $\mathfrak{G}$ consists of at least one directed graph, there should be a path in the tree that has length E. Now, one could think of running Algorithm 2.2.1 as moving a particle along the tree starting at the root until a vertex of depth E is reached.

Applying the ”Go with the Winners” idea would mean to start multiple instances of Algorithm 2.2.1 to generate a colony of digraphs. It is very well possible that some of them get stuck because of an attempt to create a self- or a multiple edge. Now, we would not want to restart them, but clone some of the rest instead. This action will restore the size of the colony, so there are enough configurations to continue with. The following algorithm describes this procedure:

Algorithm 2.3.1.

1. *Create a colony of B empty directed graphs. For each directed graph in it assign a proper number of incoming and outgoing stubs to its nodes according to the desired degree sequence.*

2. *Start to connect the incoming and outgoing stubs like in the unmodified*

version of the matching algorithm. Remove configurations where an attempt to create a self- or multiple edge has been made.

3. *Monitor the size of the colony and if it drops under $B/2$, clone the rest until there are again B configurations that can continue. Let c be the number of such cloning events.*

4. *If the size of the colony drops to 0 at some point, go back to step 1. Otherwise wait until all configurations are completed, choose one of them at random and output it.*

Now, we have an algorithm that generates random digraphs, so we have to answer the following two questions: does it generate them uniformly and how much time is needed for this.

As expected, the answer to the first question is negative. If we just take the output, it will not be uniformly sampled from $\mathfrak{G}$. This stems from the fact that the method gives credit to some intermediate configurations because of the cloning events. Therefore, one has to correct for this bias and one way to do it properly is to use weighted samples. The authors of [17] propose to set the sample weight to 2^{-c}, where c is the number of cloning events. In [15] the weight was set to $2^{-c}\frac{b}{B}$, where c is the number of cloning events and b is the number of directed graphs at the end of the process of connecting stubs. The latter was implemented also in the network motif detection software *mfinder* [22]. Both choices reflect the fact that each cloning event doubles the amount of configurations to continue with, but does not add any diversity at this point. Although this issue deserves deeper investigation, we follow the same line of argumentation and set the sample weight to 2^{-c}. Suppose we run Algorithm 2.3.1 Q times and we want to estimate the mean of some quantity, say X_i, in its outputs, $G_i \in \mathfrak{G}$ for $1 \geq i \geq Q$. Let $W_i = 2^{-c_i}$ be the weight for the i^{th} output. Then, the estimated mean of X in $\mathfrak{G}$ would be

$$\frac{\sum\limits_{i=1}^{Q} W_i X_i}{\sum\limits_{i=1}^{Q} W_i}.$$

In the case of motif finding, X_i will be the number of occurrences of a particular subgraph in G_i. In this way one can compute the averages correctly and estimate the statistical significance of the findings.

The question of how much time will Algorithm 2.3.1 need to come to completion is harder. Obviously, it is bound by the time that the unmodified version of the matching algorithm needs divided by B, but we will try to analyze it more precisely. In [26] Aldous and Vazirani introduce several concepts in order to do so. Following their paper, we will define $p(v)$ to be the probability that a particle according to the unmodified version of the matching algorithm visits the vertex v in the decision tree. Let $a(j) = \sum_{v \in V_j} p(v)$, where V_j is the set of all vertices at depth j, be the probability that a particle reaches at least depth j. Let $p(w|v)$ and $a(j|v) = \sum_{w \in V_j} p(w|v)$ be the probability to reach vertex w and depth j respectively conditioned on being at vertex v. For $i < j$ let

$$\kappa_{i,j} = \frac{a(i)}{a^2(j)} \sum_{v \in V_i} p(v) a^2(j|v)$$

and

$$\kappa = \max_{0 \leq i \leq j \leq E} \kappa_{i,j}.$$

Let d be the desired depth (in our case $d = E$).

The most important parameters in this analysis will be $\kappa_{i,j}$ and κ respectively, so we will take a closer look at them. $\kappa_{i,j}$ can be rewritten as follows:

$$\begin{aligned}
\kappa_{i,j} - 1 &= \frac{a(i)}{a^2(j)} \sum_{v \in V_i} p(v) a^2(j|v) - \frac{a^2(j)}{a^2(j)} \\
&= \frac{\frac{\sum\limits_{v \in V_i} p(v) a^2(j|v)}{a(i)} - \frac{a^2(j)}{a^2(i)}}{\frac{a^2(j)}{a^2(i)}} \\
&= \frac{\sum\limits_{v \in V_i} p(v|i) a^2(j|v) - \left(\frac{\sum\limits_{v \in V_i} p(v) a(j|v)}{a(i)} \right)^2}{\left(\frac{\sum\limits_{v \in V_i} p(v) a(j|v)}{a(i)} \right)^2} \\
&= \frac{\sum\limits_{v \in V_i} p(v|i) a^2(j|v) - \left(\sum\limits_{v \in V_i} p(v|i) a(j|v) \right)^2}{\left(\sum\limits_{v \in V_i} p(v|i) a(j|v) \right)^2} \\
&= \frac{Exp\left(a^2(j|W_i)\right) - \left(Exp\left(a(j|W_i)\right)\right)^2}{\left(Exp\left(a(j|W_i)\right)\right)^2} \\
&= \frac{Var\left(a(j|W_i)\right)}{\left(Exp\left(a(j|W_i)\right)\right)^2} \geq 0,
\end{aligned}$$

where W_i is the position of a particle at depth i and $Exp(X)$ and $Var(X)$ are the expected value and the variance of the random variable X. From the above it follows that both $\kappa_{i,j}$ and κ are at least 1. Also, one could see their interpretation. $\kappa_{i,j} - 1$ is the square of the coefficient of variance of the random variable $a(j|W_i)$, which gives the probability to reach depth j if the algorithm is at node W_i. Since κ is the maximum of all these values, it gives the upper bound for their variability. Therefore, one can interpret it as a measure of the imbalance of the tree, which is also intuitively crucial for the success rate of the algorithm. In the case of $\kappa = 1$, every path in the tree has length d. On the other hand, there is no upper bound for κ, since one can let the variance of $a(j|W_i)$ grow infinitely while holding its expected value constant.

Furthermore, Aldous and Vazirani state the following theorem under the assumption that for each i, $\frac{a(i)}{a(i+1)} \geq poly(d)$, where $poly(d)$ is a polynomial of d.

Theorem 2.3.1. *There is a polynomial poly() such that if Algorithm 2.3.1 is run with $B = \kappa \times poly(d)$, it fails to find a leaf of depth d with probability at most 1/4.*

The reader can find the proof of it in [26]. For our purposes it is sufficient to say, that if one could bound κ by a polynomial or a constant, the whole algorithm runs in polynomial time with a constant probability.

Unfortunately, calculating κ for a given degree sequence is also very complicated. It depends, for example, on the number of all possible directed graphs in $\mathfrak{G}$, which, as we stated, is still hard to find out exactly. It is also not very clear how to calculate the number of children for each non-leaf node in order to compute the values $a(i)$ and $a(i|W_i)$, respectively. Therefore, one is forced again to set some fixed number of particles, B. An alternative would be to let B be proportional to the size of the input, say n or E.

3. RESULTS

We implemented all three algorithms from the last section including their modifications in C++ and a parallel version of the "Go with the Winners" using the Message Passing Interface(MPI).

The sequential algorithms have been run on a linux workstation with one 1GHz Pentium processor and 256Mb RAM. The parallel version of "Go with the Winners" have been run on a LAM/MPI [5, 24] cluster.

We used the degree sequence from the network of transcriptional regulation in E.coli as input to test the algorithms (see Figure 3.1). Each vertex in it represents a gene. An edge between two vertices is drawn if there is experimental evidence that the source is a transcription factor that regulates the target gene. This information was collected from RegulonDB [21] and some additional literature by the authors of [22]. We used also some toy-size degree sequences, where we could calculate the cardinality of $\mathfrak{G}$ in a brute force way. The comparison of running times using the degree sequence from the E.coli network is given in Table 3.1.

3.1 The Switching Algorithm

The switching algorithm shows good performance, because it needs $O(E)$ time to come to completion. Using inputs of small size we observed that $\tau = E$ is sufficient for good sampling. The authors of [15] performed a simulation with the network of transcriptional regulation in E.coli and monitored the concentration of feedforward loops while increasing τ. Their results show that it converges after around E steps to some value, that seems to be the average number for directed graphs with this degree sequence. Therefore, they are hypothesizing that $\tau = 100 * E$ should be more than adequate in practice. We run our implementation with this setting, but did not analyze if the samples are drawn uniformly from $\mathfrak{G}$. The algorithm produces a sample of size 100 of simple directed graphs with the same degree sequence as the E.coli network in 7 seconds (see Table 3.1).

Algorithm	Parameters	Statistics
Switching	$\tau = 100 * E$	Time:7 sec
Matching unmodified	-	restarts:29000 time:2h 24min 50sec
Matching modified	$\varepsilon = 1000$	restarts:0.3 retries:11.9 time:6 sec

Go with the Winners	$B = 100$
	processors = 20
	sync_steps = 10
		restarts:0
		cloning_steps:17.3
		time:1min 48sec

	$B = 100$
	processors = 2
	sync_steps = 10
		restarts:0
		cloning_steps:14.7
		time:7min 53sec

Tab. 3.1: Comparison of the running times using the degree sequence from the network of transcriptional regulation in E.coli as input. Each run generated 100 random directed graphs. **restarts** is the number of times that an algorithm discarded a partial configuration and started again from scratch. **retries** is the number of times that the modified version of the matching algorithm tried to correct an attempt to create a self- or a multiple edge. **cloning_steps** is the number of cloning events during the Go with the Winners algorithm. These three values are averaged over the whole sample. **processors** is the number of processors used for the parallel version of Go with the Winners. **sync_steps** is the rate at which the worker processes were synchronized.

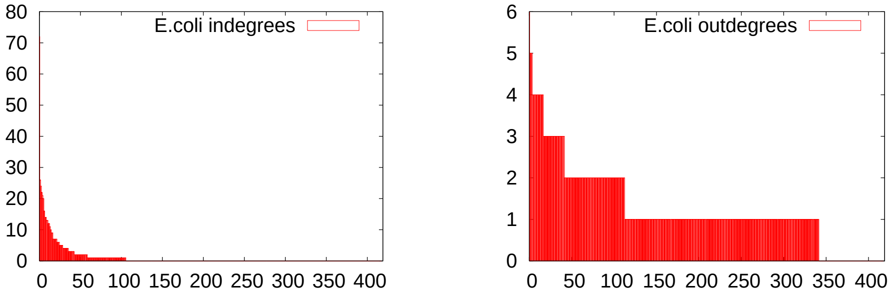

Fig. 3.1: Statistics on the network of transcriptional regulation in E.coli. The X-axis denotes the labels of the vertices if sorted by indegree(left) or outdegree(right) in descending order. The Y-axis denotes the in- or the outdegree respectively. The graph contains 418 vertices and 519 edges with average of 1.2 edges per node. The maximum indegree is 72 and the maximum outdegree - 6.

3.2 The Matching Algorithm

The unmodified version of the matching algorithm is as expected the slowest algorithm from all. Again, it performs relatively good on toy-size inputs, but using it to generate samples for background models for real data (such as the E.coli network) would clearly take more time that one can afford (see Table 3.1).

The modified version of the matching algorithm runs in contrast much faster. Nevertheless, it is hard to estimate analytically how biased the generated samples are. Therefore, we analyzed them empirically. Although the discrepancy between P_a and P_b in the example of Figure 2.7 is only $\frac{5}{432}$ and does not seem to be significant, there are cases where the bias is considerably more tangible. If we let $\varepsilon = 1000$ and take the following degree sequence-

$$\langle(0,10),(1,1),(1,1),(1,1),(1,1),(1,1),(1,1),(1,1),(1,1),(1,1),(1,1),(10,0)\rangle$$

and its realizations (Figure 3.2), we will see that the digraph in Figure 3.2a) is drastically undersampled. This degree sequence is derived from the previous example by multiplying the number of internal nodes (2-10 in Figure 3.2). In this way node 1 becomes an outgoing hub and node 12 an incoming hub. There are two distinct topologies - both hubs are connected to all of the internal nodes (Figure 3.2a) or there is one edge between two internal nodes (Figure 3.2b). There are 90 possible digraphs from the second type and

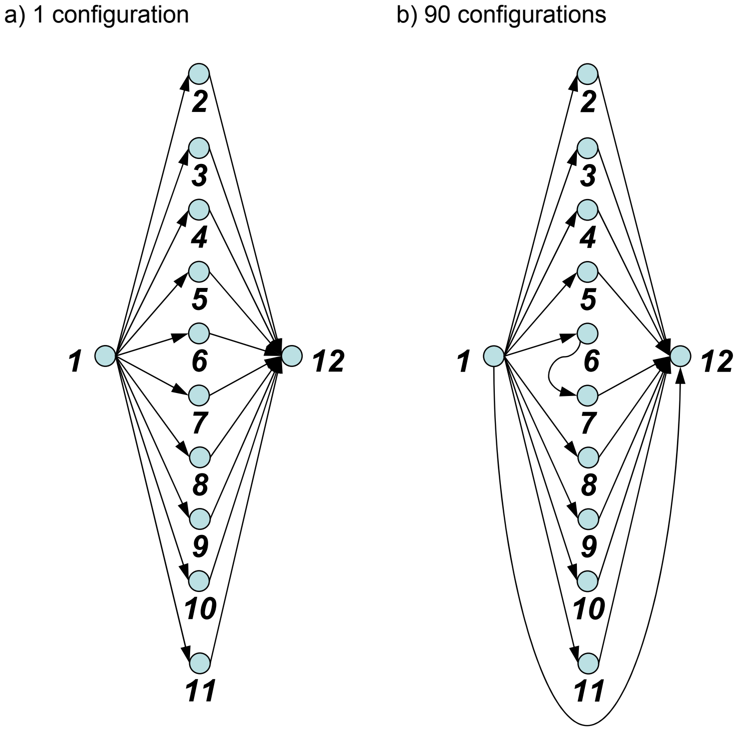

Fig. 3.2: Example of biased sampling of the modified version of the matching algorithm

ε	Figure 3.2a	Figure 3.2b	Retries Mean	Retries STD
0	1078	~ 1100	~ 2000	~ 2000
1	516	~ 1100	~ 650	~ 650
2	289	~ 1100	~ 320	~ 320
1000	53	~ 1100	~ 67	~ 67

Tab. 3.2: Statistics for the generation process of the 91 different graphs from Figure 3.2 with Algorithm 2.2.3. The counts for Figure 3.2b) are averaged over all 90 configurations. The mean and the standard deviation of the number of retries are averaged over all 91 configurations.

because they are all isomorph, their probabilities will be equal even with the modified version of the matching algorithm. To calculate them and the probability of Figure 3.2a) exactly will be very time and space consuming due to the combinatorial complexity. Therefore, we examined empirically their frequencies in a sample of 100000 runs of Algorithm 2.2.3. The digraph of Figure 3.2a) was generated 53 times, where each of the other 90 was generated 1100 times on average. We measured the number of times that both topologies were generated on average for different values of ε, the mean number of retries and its standard deviation (Table 3.2). The number of retries does not differ significantly if the final result is a digraph of type Figure 3.2a) or Figure 3.2b) and is geometrically distributed with mean given in Equation 2.5 for $\varepsilon = 0$. For $\varepsilon > 0$ the mean number of retries decreases as expected, but is hard enough to calculate it analytically.

Our results are confirmed also by the simulations performed in [15] and clearly show that the bias could become significant for specific degree sequences.

3.3 The "Go with the Winners" Algorithm

The parallel implementation of the "Go with the Winners" algorithm distributes the particles evenly among the nodes of the cluster. Generally, the processes on the cluster are divided into a single coordinator process and multiple workers. Each worker receives a subset of configurations (called *minicolony*) that are computed sequentially then. If their number drops under the half of the initial size of the minicolony, then all of the remaining configurations are cloned until the size of the minicolony grows to a number that is more than the half of its initial value. Furthermore, it is still possible that a worker discards all of its configurations due to some attempts to create self- or multiple edges. In this case it sends a message to the coordinator and waits to be synchronized with another worker. The coordinator performs synchronization steps at a predefined constant rate (for example every 10 steps). It waits for all the workers to report their current progress and then decides upon the strategy of the load balancing. In our implementation we used a simple one. We divide the nodes into two categories: *alive* and *dead*, according to whether they have some configurations available at the current step or not. If there are more dead than alive nodes, we assign to each alive node a number of dead nodes and require from it to send them its whole minicolony. In this way one actually clones the whole worker process, but this still has the same effect.

The parallel version of the "Go with the Winners" runs also quite fast. A run on 20 nodes with $B = 100$ generates a single directed graph with the same degree sequence as the E.Coli network in around one second on average.

4. DISCUSSION

The problem of random generation of graphs is far from exhausted with this work. Many questions remain open. For example, one could notice that the rigorous analysis of the time complexity of all three algorithms stops at the point where one needs to know the exact number of all possible directed graphs with this degree sequence. It is hard to say anything about the convergence rate of the Markov chain. The probability for the matching algorithm to come to completion is also hard to calculate and the number of particles that are needed at the beginning of the "Go with the Winners" algorithm is undetermined for the same reason. Although there is a series of papers [1, 2, 11, 23] trying to resolve this issue, the best known result to our knowledge is an approximation for N that is true for a fixed upper bound on the in- and outdegrees and $n \to \infty$. In our toy examples it did not predict even close to the actual number of digraphs. Therefore, it is hard to tell how good it may be if used in real situations.

This brings our attention to the second problem: how many directed graphs does one have to generate in order to be confident when making a statistical statement? If the space $\mathfrak{G}$ contains billions of billions representatives and the size of the sample generated by the algorithm is only some couple of thousands, it may be very well possible that one draws wrong conclusions about the significance of the quantities that are measured in the original data. One should think of some way to evaluate the robustness of the algorithms used for the background model.

A third problem is how to estimate the bias of the modified version of the matching algorithm. It is clear that the more retries are allowed the more tangible it becomes. The authors of the *mfinder* software [15, 22] use this algorithm for the background model that assesses the statistical significance of the network motifs in various graphs and claim that the bias is acceptable, but in our opinion it needs more rigorous analysis.

A similar problem occurs when one decides how to set the sample weights for the "Go with the Winners" algorithm. It is not obvious why 2^{-c} is a correct choice. According to the theory of importance sampling [4], in any sampling procedure where one makes a decision with probability p in order to obtain an unbiased sample, one could replace p with p' and correct for this by setting the sample weight to $\frac{p}{p'}$. In our opinion it is not clear that in the case of generation of random graphs with the "Go with the Winners" algorithm why $\frac{p}{p'} = 2^{-c}$. We leave this issue as an open question.

It is worth noting that parallel versions of "Go with the Winners" algorithms exist already for some problems [18]. The structure of the parallel architecture also imposes some computational burden. For example, the nodes have to be synchronized from time to time via messages. To analyze rigorously this overhead is also tricky since it requires some knowledge about the topology of the decision tree of the matching algorithm. Thus, we leave it as an open question.

Yet another problem that surely deserves attention is the one of generating isomorphic graphs. None of the algorithms described in this work deal with it. The problem is that the space $\mathfrak{G}$ contains classes of isomorphic graphs and their number is hard to calculate since it depends on the cardinality of $\mathfrak{G}$. Even worse, if one knows this cardinality, to calculate the number and the sizes of the isomorphic classes requires the usage of the Polya's Theory of Counting [19], which makes it quite complicated not only from the analytical but also from the practical side. Very often to evaluate a formula derived by this means takes more time than to enumerate all possibilities in a brute-force way and just do the counts.

It is easily seen why isomorphic classes cause a problem. Depending on the degree sequence not all of them might be of size one. In fact, for degree sequences from real networks it is highly unlikely to be so. Thus, if one starts to sample from $\mathfrak{G}$ without correcting for this, the results will be biased. To the best of our knowledge, no one in the field of network motifs discovery had payed attention to this problem.

To conclude the discussion we should say that generating random directed graphs with the same degree sequence as the original network may not be always an appropriate null hypothesis. In many cases it is questionable if this property actually reflects the natural constraints that govern the world that has been modeled. One could think of alternative ways to describe the background distribution. For example, by generating random directed graphs that have a certain minimal or maximal path lengths, number of cycles, girth, concentrations of subgraphs, etc. It is very well possible that such requirements complicate the synthesis and analysis of the algorithms that sample from these spaces and therefore will be a good challenge from computer science point of view. Another approach would be to compile a finite set of production rules for directed graphs called "graph grammar". Then one could implement a probabilistic algorithm that generates words from the language defined by this grammar and analyze their properties. The crucial issue in this case is how to choose the production rules and their probabilities so that they reflect the phenomena from the field of interest.

All of this and surely more challenging questions will be in our opinion subject of the future work in this area.

5. SUMMARY

In this work we analyzed three algorithms for uniform generation of directed graphs with prescribed degree sequence: the switching, the matching and "Go with the Winners" algorithms. Each of them has its advantages and drawbacks. The switching algorithm is correct in the limits and quite fast in practice because it runs in linear time and space, but one cannot tell if the output is drawn from the stationary distribution. The matching algorithm is correct also but comes to completion with very low probability for degree sequences derived from real data. Its modified version runs in contrast much faster but produces biased samples that are complicated to analyze. The "Go with the Winners" algorithm samples also non-uniformly but it is possible to correct it by assigning appropriate weights to the output. Its parallel version runs also quite fast and is therefore a good alternative to the other two. To give a straightforward criterium which one to choose is hard and probably application-dependant. In our opinion, to make the right decision requires more evaluation and analysis in the future.

6. ACKNOWLEDGEMENTS

First of all, I am very grateful to Prof. Martin Vingron for supervising my thesis. It was a pleasure to work with him and I have learnt many new things during this time.

Furthermore, I would like to thank to Debora Marks from the Harvard Medical School for bringing our attention to the problem of network motif detection in the context of our work on micro-RNAs. I would like to thank to Dr. Behshad Behzadi for his critical review of this manuscript and for pointing and discussing with me a great deal of issues that stemmed from it. Also, I am very thankful to Dr. Clemens Gröpl for his help as a second reviewer of my thesis. I would like to thank to Dr. Stephan Röpcke for discussing with me many of the problems during my work. I would like to thank to Todor Koev for reading the manuscript and improving the quality of the text and to Boyko Amarov for the endless nights he spent to talk with me about probabilities and statistics. I really appreciate their help. I would like to thank to all my friends and especially to my parents, Plamen and Tsvetanka Fizievi, for their continuous support along my study and this work in particular.

Last but not least, I would like to thank to the Max Planck Institute for Molecular Genetics for the financial support that it would surely be much more difficult to complete my diploma thesis without.

BIBLIOGRAPHY

[1] A.Békéssy, P.Békéssy, and J.Komlós. Asymptotic enumeration of regular matrices. *Studia Scientiarum Mathematicarum Hungarica*, 7:343–353, 1972.

[2] E.A. Bender. The asymptotic number of non-negative integer matrices with given row and column sums. *Discrete Mathematics*, 10:217–223, 1974.

[3] B. Bollobas. *Random Graphs.* Cambridge University Press, 2001.

[4] J.A. Bucklew. *Introduction to Rare Event Simulation.* Springer-Verlag, New York, 2004.

[5] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[6] P. Erdos and A. Renyi. On Random Graphs I. *Publ. Math. Debrecen*, 6(290), 1959.

[7] W. Feller. *An Introduction to Probability Theory and its Applications.* Wiley, New York, 1968.

[8] S. Itzkovitz and U. Alon. Subgraphs and network motifs in geometric networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 71(2 Pt 2):026117, 2005. 1539-3755 Journal Article.

[9] S. Itzkovitz, R. Milo, N. Kashtan, G. Ziv, and U. Alon. Subgraphs in random networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 68(2 Pt 2):026127, 2003. 1539-3755 Journal Article.

[10] R. Kannan, P. Tetali, and S. Vempala. Simple Markov-Chain algorithms for generating bipartite graphs and tournaments. In *Symposium on Discrete Algorithms*, pages 193–2000, 1997.

[11] L. Katz and J.H. Powell. The number of locally restricted directed graphs. *Proceedings of the American Mathematical Society*, 5:621–626, 1954.

[12] M. Krause. A Simple Proof of the Gale-Ryser Theorem. *Amer. Math. Monthly*, 103:335–337, 1996.

[13] N. Metropolis and S. Ulam. The Monte Carlo Method. *J. Amer. Stat. Assoc. 44, 335-341*, 1949.

[14] R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–42, 2004. 1095-9203 Journal Article.

[15] R. Milo, N. Kashtan, S. Itzkovitz, M.E.J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *arXiv:cond-mat/0312028*, 2004.

[16] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–7, 2002. 1095-9203 Journal Article.

[17] W. Nadler and P. Grassberger. "Go with the Winners" simulations. *preprint cond-mat/0010265*, 2000.

[18] M. Peinado and T. Lengauer. Parallel "Go with the Winners Algorithms" in the LogP Model. *IPPS: 11th International Parallel Processing Symposium*, 1997.

[19] G. Polya, R. Tarjan, and D. Woods. *Notes on Introductory Combinatorics,Progress in Computer Science No. 4*. Birkhauser, Boston, Basel, Stuttgart, 1983.

[20] A.R. Rao, R. Jana, and S. Bandyopadhyay. A Markov Chain Monte Carlo Method for Generating Random (0,1)-Matrices. *The Indian Journal of Statistics*, 58:225–242, 1996.

[21] H. Salgado, A. Santos-Zavaleta, S. Gama-Castro, D. Millan-Zarate, E. Diaz-Peredo, F. Sanchez-Solano, E. Perez-Rueda, C. Bonavides-Martinez, and Collado-Vides J. RegulonDB (version 3.2): transcriptional regulation and operon organization in Escherichia coli K-12. *Nucleic Acids Res.*, 29(1):72–74, 2001.

[22] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of Escherichia coli. *Nat Genet*, 31(1):64–8, 2002. 1061-4036 Journal Article.

[23] T. Snijders. Enumeration and simulation methods for 0-1 matrices with given marginals. *Psychometrika*, 56:397–417, 1991.

[24] J.M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. *Proceedings, 10th European PVM/MPI Users' Group Meeting*, 2840:379–387, September / October 2003.

[25] R. Taylor. Constrained switchings in graphs. *Lecture Notes in Mathematics*, 884:314–336, 1981.

[26] U.V. Vazirani and D. Aldous. "Go With the Winners" Algorithm. *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages pp. 492–501, 1994.

[27] F. Viger and M. Latapy. Fast generation of random connected graphs with prescribed degrees. *arXiv:cs.NI/0502085*, 2005.