Probabilistic k-swap method for uniform graph generation beyond the configuration model

Lionel Tabourier, Julien Karadayi

Probabilistic k-swap method for uniform graph generation beyond the configuration model

Lionel Tabourier and Julien Karadayi

Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Abstract

Generating graphs with realistic structural characteristics is an important challenge for complex networks analysis, as these graphs are the null models that allow to describe and understand the properties of real-world networks. However, the field lacks systematic means to generate samples of graphs with predefined structural properties, because it is difficult to devise a method that is both flexible and guarantees to get a uniform sample, *i.e.*, where any graph of the target set has the same probability to be represented in the sample. In practice, it limits the experimental investigation to a handful of models, including the well-known Erdős-Rényi graphs or the configuration model. The aim of this paper is to provide such a method: we design and implement a Monte-Carlo Markov Chain process which is both flexible and satisfies the uniformity condition. Its assumptions are that: 1) the graphs are simple, 2) their degree sequence is fixed, 3) the user has at least one graph of the set available. Within these limitations, we prove that it is possible to generate a uniform sample of any set of such graphs. We provide an implementation in python and extensive experiments to show that this method is practically operational in several relevant cases. We use it with five specific set of constraints and verify that the samples obtained are consistent with existing methods when such a method is available. In those cases, we report that state-of-the-art methods are usually faster, as our method favors versatility at the cost of a lower efficiency. Also, the implementation provided has been designed so that users may adapt it to relevant constraints for their own field of work.

Keywords— graph generation, uniform sampling, Monte Carlo Markov Chain method, conditionally uniform graph models

1 Introduction

Describing the structure of a complex network as a graph is a delicate question, because it demands a comparison to some form of benchmark that allows to decide if an observed property is expected or not. This is why *null models generation* has been an important focus of interest since the early days of network science. An appropriate null model may even provide an explanation to the structure of the network in the sense that it reduces

its structural complexity to a few features. For example, the field of social network analysis introduced models long before the advent of online social networks to explain the observed structure, as for instance in [48] or [19]. Beyond this explanatory function, null models also provide artificial networks with specific characteristics that resemble the ones of real-world networks when data is missing, which is of utmost importance for simulation purposes.

To serve their function as null models, the generation process should have no or limited bias relatively to characteristics which are not contained in the model. A controlled way of doing so is to generate uniform samples: any graph in the target set of graphs has strictly the same probability to be in the sample. Unfortunately, uniform generation models are hard to design. Whereas it is often possible to generate a graph having specific properties, it is much more difficult to create an unbiased sample of such graphs.

In this work, we propose a method in order to generate null models in a versatile way. More precisely, our contributions are the following: first, we propose a Markov Chain based method and prove that it is capable of generating uniform samples of graphs satisfying any set of properties that includes the degree sequence. It is a significant step forward as methods in the literature are either uniform but limited to specific properties, or statistically biased, which weakens the interpretation of their results. Second, we adapt to our problem an experimental process that evaluates the convergence of this Markov Chain based method. Third, we provide a python implementation of the method in several cases which are commonly investigated models in the literature and check experimentally that the method can be used in practice.

In Section 2, we provide an overview of the works on generating graphs uniformly at random on practical instances. Then, in Section 3, we describe the method proposed and prove that it satisfies all the necessary criteria to provide a uniform sampling of the target set of graphs. Finally, in Section 4, we give a detailed experimental report of the investigation achieved with the implementation of the method provided with this work. The implementation is not limited to the realization of the Markov process itself, but also includes an automated experimental test that the process has converged toward a uniform sample. We also discuss in that section the practical limits of the method: making it adaptable to new problems and automatizing the convergence test has a computational cost, which is why specialized methods tend to outperfom it on specific target set of graphs.

2 State of the art

In this section, we discuss models in the literature proposed for comparison to real-world networks. As a comprehensive review of this question is beyond the scope of our work, we focus on models that provide homogeneous samples of a set of graphs with a given set of properties, that we call respectively *target set* and *target properties*. This category of models are sometimes referred to as *conditionally uniform graph models* [40]. Moreover, we focus on vertex-labeled graphs, meaning that nodes can be distinguished from each other.

2.1 Historical models based on constructive methods

The best-known conditionally uniform graph model is without doubt the Erdős-Rényi model [8]. According to it, the target properties are the number of nodes n and edges m, or alternatively the number of nodes and a fixed density (the density δ being here defined

as the number of existing edges divided by the number of possible edges). However, this model has limited use for comparison to real-world networks, because the structure of the graphs obtained differs in many significant characteristics from real-world ones: degree distribution, local density, etc.

Graphs with a given degree sequence are widely used models for comparison to real-world networks. The *configuration model* (see for instance [32]) refers to a standard strategy to build a random graph with a given degree sequence. It consists essentially of giving to each node as many stubs – i.e., half-edges – as its degree, and then pairing stubs randomly. As long as the degree sequence is graphical, this method allows to generate uniformly random multi-graphs (allowing multiple edges and self-loops). More precisely, it generates uniformly at random stub-labeled multi-graphs, as discussed in details in [15]. When considering simple graphs, generating uniformly stub-labeled graphs is equivalent to generating uniformly vertex-labeled graphs. However, the pairing process naturally leads to the creation of multi-edges and self-loops. A basic way to solve this issue is to restart the generation process whenever a multi-edge or a self-loop is created. But this may lead to extremely high failure rates, especially with real-world graphs, which often exhibit a heavy-tailed degree distribution. Improved polynomial algorithms have been proposed to produce random graphs with a given degree sequence provided that the sequence does not exhibit too many high degree nodes [5, 2]. Note also that these methods come at the cost of much more elaborate algorithms. Another way to circumvent the limitations of the configuration model consists in producing samples which are not uniform, but the bias of which is known. Then, it is possible to correct the measurements on the biased sample to simulate what would be the corresponding measure with a uniform sample. These methods are usually known as *importance sampling methods*, and works such as [12] and [7] propose methods of this kind.

2.2 On Monte Carlo Markov Chain methods

The methods described above to generate graphs with a given degree sequence are *constructive methods* as the graph is created according to a given building process which uses the target properties as an input. On another note, *Monte Carlo Markov Chain* (denoted MCMC) methods start from any element of the target set and then apply elementary modifications iteratively until loosing memory of the input graph to obtain a random element of the target set.

Desired properties of MCMC methods. General MCMC methods for sampling or enumerating ensembles have been popularized during the 80's and 90's in particular by the works of Jerrum and his colleagues [21]. In order to actually reach a random element of the set with a uniform probability, we need the Markov chain to satisfy the three following properties:

- *irreducible* (or ergodic): any element can be reached from any other element,
- *positive recurrent*: the expected return time from any element to itself is finite,
- *aperiodic*: all its states are aperiodic, i.e., there is no $T > 1$ so that the chain can return to an element only after a number of steps which is a multiple of T.

Edge-swap based MCMC methods. Several instances of these processes have been explored on graphs, one of the most popular is certainly the *edge swap* (also called *edge switch*, or *rewiring*, or *flip*) which consists in selecting two random edges and exchanging their ends. There are variants of this basic description, depending on the fact that

the graph is directed or not, a simple, multi or pseudograph. This method is commonly used to generate undirected simple graphs with a given degree sequence by iterating edge swaps until it leads to a random element (e.g., [27, 3]), the process being irreducible [44]. More precisely, suppose that edges (u, v) and (x, y) have been randomly drawn, a standard edge swap replaces these two edges by (u, x) and (v, y). Such a swap is authorized if and only if edges (u, x) and (v, y) do not already exist, u and x are distinct nodes, as well as v and y, thus avoiding the creation of multi-edges or self-loops. Another way of looking at this situation is that an alternating cycle of length four has been found in the graph. We remind that an alternating cycle is an even sequence of edges alternating with non-edges of the form $(u_1, u_2), \overline{(u_2, u_3)}, \ldots, \overline{(u_{k-1}, u_k)}, (u_k, u_1)$ where $\overline{(u_i, u_j)}$ denotes a non-edge, as represented in Figure 1. The corresponding adjacency matrix configuration, called a *tetrad* is represented opposite to the alternating 4-cycle. Here, the swap itself consists in changing the edges of this cycle into non-edges and vice versa.

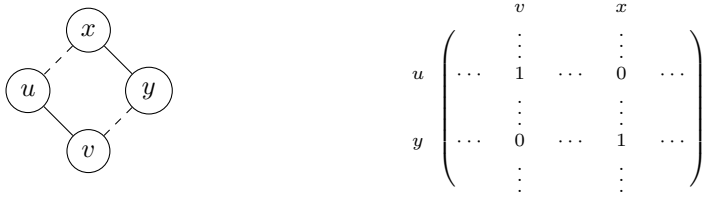

Figure 1: Left: alternating 4-cycle, plain lines correspond to existing edges, dotted lines correspond to non-existing edges. Right: corresponding tetrad configuration in the adjacency matrix (for undirected graphs, a symmetric tetrad also exists).

Any discrete markovian process can be represented as a directed graph, called Markov graph. Its nodes are the elements (i.e., graphs) of the target set to be described, and directed edges represent the possibility to go from one element to another through an elementary modification. The directed edges of the Markov graph are weighted by the probability to go from a node to another according to a given process. The irreducibility of the Markov chain translates into having a strongly connected Markov graph.

A fundamental issue with these methods is that the number of elementary modifications needed to reach the stationary state (or *mixing time*) of the Markovian process is unknown in general. Determining the cases where the mixing is rapid (which is defined as having a mixing time which is a polynomial function of the logarithm of the number of steps in the chain) has been a focus of much work [20, 38]. In the case of graphs with a given degree sequence, the works of Péter Erdős *et al.* [14] addressed the problem by establishing graph properties which allow to have rapid mixing. While proving the rapid mixing of MCMC methods is often a difficult task, it has been observed experimentally that the edge-swapping MCMC to produce graphs with a given degree sequence has in general a short enough mixing time to be used practically [17, 28, 47, 13], making it an interesting method to generate unbiased samples of such graphs.

Beyond edge-swap based MCMC methods. Rao et al. [35] successfully applied MCMC methods to the uniform generation of directed simple graphs with a given degree sequence. This problem is relevant as a simple swap method is known to be reducible and thus cannot be used to uniformly generate samples of such graphs. However, the

authors prove that it is possible to create an irreducible Markov process in the directed case too. Using an adjacency matrix description, a simple directed graph is represented by a boolean matrix which must contain 0 in its diagonal. With such a description, a swap is an alternating 4-cycle (or tetrade), as represented in Figure 1. The authors achieve irreducibility by using not only alternating 4-cycles but also alternating 6-cycles in the MCMC process. An alternating 6-cycle is an extension of the alternating 4-cycle equivalent to finding an *hexade* when considering adjacency matrices (see Figure 2).

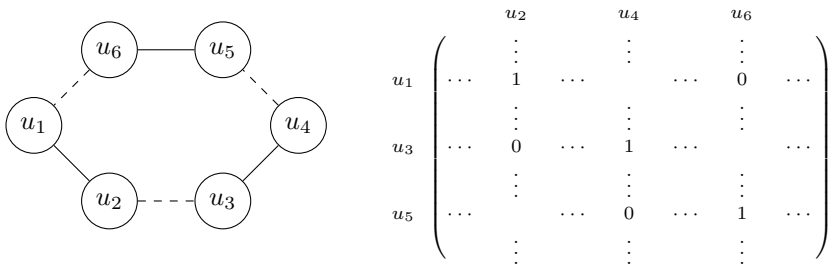

Figure 2: Left: alternating 6-cycle, plain lines correspond to existing edges, dotted lines correspond to non-existing edges. Right: corresponding hexad configuration in the adjacency matrix (for undirected graphs, a symmetric hexad also exists).

Among the MCMC methods designed to generate graphs with a given degree sequence, it is also worth mentioning Carstens *et al.*'s `Curveball` and `Global Curveball` methods [9], which are able to randomize uniformly graphs with a given degree sequence in multiple contexts: bipartite graphs, undirected and directed graphs with self-loops. To achieve this, the authors use a specific kind of elementary step called *trades*, previously introduced by Verhelst [46], without going into more details, a trade modifies more edges of the graph than a basic swap does, which explains higher convergence speed in practice.

2.3 Beyond graphs with a given degree sequence

In spite of their successes, graphs with a given degree sequence cannot account for local density, degree correlations, community structure, etc. Thus, an important effort has been made to go beyond this model and propose new models and methods that yield uniform samples of more realistic and elaborate constraints.

Models including degree correlations. Generating graphs with a given degree sequence and given degree correlations has attracted interest in various ways. Degree correlation may indeed be integrated to the model in a constricting form by setting the joint degree matrix (JDM) of the graph, that is to say the number of edges connecting a node of degree k to a node of degree k' [41, 16, 4, 10, 1]. It can also be added to the model with less strict constraints, for instance by setting the Pearson correlation coefficient between degrees of neighbors in the graph [30, 11]. Some authors have proposed the dK-series framework [25] where degree-based graph models are part of a hierarchical ensemble: the 1K-model corresponds to random graphs with a given degree sequence, the 2K-model

corresponds to random graphs with a given joint degree distribution, etc. While dK-series were initially considered in the context of the Internet topology, this framework has also been used for other applications [34]. However, it is unclear in [25] how the generation processes actually work for $d \geq 2$, besides that, there is currently no standard method to generate dK-graphs for $d > 2$.

The earliest models to generate graphs with degree correlations were known to be biased [30, 11] and they were used to investigate the effect of degree correlations on other phenomena, such as epidemic spreading. Gjoka *et al.* [16] propose to generate graphs with a prescribed degree sequence and additional properties: a fixed joint degree matrix (JDM) and possibly other properties such as a tunable clustering coefficient or specific node attributes. However, the generating process proposed is not uniform. To the best of our knowledge, Stanton and Pinar [41] are the first to propose a method to generate a random graph with a given JDM in a uniform way. First, they describe an algorithm that generates an instance of a graph having a given JDM. Second, they propose a MCMC method based on an edge-swap procedure to sample the set of graphs with the given JDM. They prove that this procedure guarantees to obtain a uniform sample, and in particular that the Markov Chain is irreducible. This proof was later shown to be flawed, but corrected proofs were proposed later to show that MCMC swap-based methods can indeed be used to generate graphs with a given joint degree matrix [10, 1]. Interestingly, the authors of [41] also highlighted the advantages and drawbacks of the MCMC based procedures: they are often the only way to generate uniform samples of graphs with elaborate constraints, however proving that they produce uniform sample is demanding. Similarly to graphs with a given degree sequence, computing a bound to the mixing time is difficult but it is observed experimentally that this time is short enough for many practical instances. Later, Bassler *et al.* [4] proposed a polynomial algorithm – precisely, quadratic for sparse graphs – to generate graphs with a given JDM. This method is exclusively constructive and offers guarantees in terms of time complexity.

Models including motifs. Another family of graphs which has been a focus of attention are random graphs with a given number of small size motifs. Here, motifs designate small connected patterns (usually 5 nodes or less), sometimes also referred to as graphlets. Triangles in particular are often thought to be a basic network motif which plays an important role in real-world networks, as suggested by the wealth of work related to the topic. Indeed, many real-world networks are known to exhibit a larger number of triangles than expected if we consider a random graph with the same degree sequence. Consequently, there have been many attempts in the literature to generate graphs with a fixed number of triangles, a fixed distribution of triangles per node, or graphs with a fixed number of various motifs [26, 31, 22, 42, 43, 33].

Constructive methods have been applied for this family too. In particular, Newman has proposed a model which is strongly related to the configuration model to generate graphs with a fixed sequence of degree and triangles per node [31]. This model has been generalized since then to random graphs with arbitrary distributions of subgraphs in [22]. While such generating processes are relatively simple to implement and analytically tractable to some extent, they have limitations which are comparable to the ones of the configuration model mentioned above.

Considering MCMC, it is known that a standard edge swap process cannot be systematically applied to guarantee uniform samples of target sets with constraints related to the number of motifs. In the context of directed graphs, it has been extensively discussed since the work by Rao *et al.* aforementioned [35]. For instance, in [37], Roberts uses an adaptation of Rao *et al.* procedure in various contexts, using alternating 4-cycles

with probability p, but also alternating 6-cycles with probability $1-p$. By this mean, he attempts at generating directed graphs with a given degree sequence and a given number of *mutual dyads*, *i.e.* links connecting two nodes in both directions. Later, McDonald [26] showed that Roberts' proposition was actually non-ergodic for the mutual dyads constraints considered. He employed related MCMC methods, which were then proved to be reducible too in the context of the target sets that McDonald considered [43].

Moreover, in the case of undirected graphs, it has been shown, for instance in [42], that the simple edge swap process is reducible for any set of graphs with a fixed degree sequence and a fixed number of triangles. In [33], Nishimura even exhibited examples of degree and triangle constraints which have disconnected Markov graphs (so, the process is reducible), even in the case of swaps involving 8 edges.

Various types of complex constraints. We now discuss other types of complex constraints which do not fall in the previous categories. In general, constructive methods have to be redefined for each specific set of properties. Consequently, for any elaborate target set, it is a new challenge to propose a constructive method that would allow to sample uniformly the target set of graphs. For this reason, MCMC methods are favored because they have the outstanding advantage of following an adaptable scheme and thus offer possibilities to achieve uniform graph generation with various sets of constraints.

In a recent article, Van Koevering *et al.* proposed a MCMC-based method to generate uniformly random graphs with a given core-sequence [45]. We remind that a k-core of a graph is its maximal subgraph in which every node has degree at least k and that this concept is most useful to describe the embeddedness of a node in the graph structure. Interestingly, the moves allowed in this MCMC method are not restricted to edge swaps: a range of *legal moves* of different sorts are possible and the authors proved that these moves guarantee the irreducibility of the Markovian process. Without going into details, the authors propose a rejection sampling trick to guarantee the uniformity of the process. Note that this contribution, by contrast with others mentioned in this section does not include the degree sequence among the target constraints.

Finally, we point at a particularly interesting method in the perspective of our study, proposed by Tao in [43]. To the best of our knowledge, it is the only instance of an algorithm in the literature that guarantees the irreducibility of a Markov process in order to generate a sample of graphs with a given degree sequence and any additional property. Instead of setting the length of the alternating cycles to swap, as is the case in several works mentioned above, the author proposes to follow an alternating path until going back to the starting node, therefore discovering an alternating cycle of variable length at each step. The author proves the irreducibility of such a process as well as its uniformity by making use of the detailed balance principle. While it is applied with some success to a few examples, notably in the cases of the target constraints of Roberts in [37] and McDonald in [26], mentioned above, the method comes at a cost: it scales rather poorly and thus can only be used on relatively small graphs. This is however an inspiring method on which we build in this work.

3 The probabilistic k-swap method

The main contribution of this work is to propose a MCMC process which is guaranteed to be irreducible, positive recurrent, aperiodic and uniform for any (non-empty) target properties of simple graphs that include the degree sequence of the graph. It improves upon the method developed in [42] which could not guarantee irreducibility and thus

resorted to a complex experimental procedure to get closer to irreducibility. In this section, we describe our method and prove that it has all the necessary properties to guarantee a uniform sampling of the target set of graphs.

3.1 Description of the Markov process

The probabilistic Markov process that we propose can be interpreted as a probabilistic k-swap. For short, we call it *pks* process in the following. At step s, we apply the following transformation to graph $G_s = (V, E_s)$:

- Draw an integer $k \in [\![2 : m]\!]$ according to the following law: $P(k) \sim \frac{1}{k^\gamma}$, γ is a parameter satisfying $\gamma \in]1 : \infty[$.
- Select k different edges randomly in E_s, arbitrarily directed and ordered: $(u_1, v_1) \dots (u_k, v_k)$.
- Let σ be a random permutation of the index in $[\![1 : k]\!]$ and consider the set of edges: $E'_s = E_s \setminus \{(u_1, v_1), \dots, (u_k, v_k)\} \cup \{(u_1, v_{\sigma(1)}), \dots, (u_k, v_{\sigma(k)})\}$, if $G'_s = (V, E'_s)$ satisfies the set of target constraints, then $G_{s+1} = G'_s$, otherwise $G_{s+1} = G_s$.

Note that the law that we use for $P(k)$ is close to the Zipf law, except for its support. It is not necessary to use the Zipf law to select k, any $P(k)$ distribution that allows to have all values of $k \in [\![2 : m]\!]$ with a non-zero probability would also ensure ergodicity. We choose it because it has a higher probability to draw low k values, which is supposed to accelerate the mixing process, as discussed in more details in Section 4.

Note also that the standard edge-swap process [27, 3] can be seen as a limit case of the *pks* process where $P(2) = 1$ and $P(k) = 0$ for any other k.

3.2 Properties of the Markov Chain

Lemma 1. *For any non-empty target set, the Markov chain is irreducible.*

Proof. For any two graphs of the target set $G_i = (V, E_i)$ and $G_j = (V, E_j)$, consider an edge (u_1, v_1) present in E_i but absent in E_j. As the degree of node v_1 is the same in E_i and E_j, there must exist an edge $(v_1, u_2) \in E_j$ and $(v_1, u_2) \notin E_i$. We can do the same reasoning to find an edge $(u_2, v_2) \in E_i$ but $(u_2, v_2) \notin E_j$ and iterate this reasoning until we reach a node u_q or v_q which is already involved in the alternating path. Without loss of generality, suppose that we find the edge $(v_p, u_1) \in E_j$ and $(v_p, u_1) \notin E_i$. The sequence $(u_1, v_1), (v_1, u_2), (u_2, v_2), \dots, (u_p, v_p), (v_p, u_1)$ is an alternating cycle between edges of E_i not in E_j and reciprocally. This reasoning may be done for any edge in $E_\Delta = E_i \Delta E_j$, the symmetric difference between E_i and E_j. Consequently, E_Δ is constituted of alternating cycles of the form given above (a similar argument is made in Tao [43]).

Also, it should be noted that $|E_\Delta| \leq m$. So, there is some $k \in [\![2 : m]\!]$ which allows to select exactly the edges in $E_i \setminus E_j$ and a permutation σ such that by applying the transformation above, we exactly find E_j. Precisely, this permutation is such that $\sigma(1) = 2, \sigma(2) = 3, \dots, \sigma(p_1) = p, \sigma(p) = 1$ for the index corresponding to the first alternating cycle, and follows a similar logic for the other alternating cycles in E_Δ. In other words, we have exhibited a transition that allows to go directly from G_i to G_j with a probability $p_{ij} > 0$. □

Note that this property is more than what we need to ensure irreducibility: we have proved that this process can connect two random graphs of the set in one step. In short, the Markov graph is complete, while we only need to have a strongly connected Markov

graph to ensure irreducibility. Still, this property guarantees to have an irreducible chain even with complicated target properties and target sets.

Lemma 2. *The Markov chain is aperiodic and positive recurrent.*

Proof. The Markov chain is finite and irreducible, thus it is positive recurrent. For any state of the chain, the probability to stay in the same state is strictly positive. Indeed, it is possible to draw for any k the permutation σ corresponding to the identity, in which case the chain remains in the same state. This ensures that the process is aperiodic. □

The Markov chain is irreducible, positive recurrent and aperiodic, so we know by theorem that it converges to a unique stationary state π.

3.3 Uniformity of the stationary state

Lemma 3. *The stationary state π of the Markov chain is uniform.*

Proof. We prove that we have detailed balance, i.e. the transition matrix of the Markov process is symmetric, $p_{ij} = p_{ji}$, which entails that the uniform distribution is the unique stationary state.

By definition, $p_{ij} = \sum_{k=2}^{m} P(k) p_{ij}^{(k)}$, where $p_{ij}^{(k)}$ is the probability to go from state i to state j for a fixed value of k and $P(k)$ is the probability to draw k. $p_{ij}^{(k)}$ is the number of permutations $N_{ij}^{(k)}$ of k edges in state i which leads to state j divided by the total possible number of k edges combinations among m edges. As for any permutation leading from state i to state j, the exact reverse permutation leads from state j to state i, we know that $N_{ij}^{(k)} = N_{ji}^{(k)}$. Moreover, the denominator only depends on k or m (and not on i or j), so we have $p_{ij}^{(k)} = p_{ji}^{(k)}$ thus $p_{ij} = p_{ji}$. □

3.4 Conclusion on the theoretical properties

From the properties proved above, we conclude that after a sufficient number of iterations of the Markov process described in Section 3.1, we obtain any graph of the target set with a uniform probability. Note that this process demands the target set to be non-empty of course, but also to have at least one element available as the starting point of the process. This constraint can often be overcome for relatively simple target sets: for instance to generate a simple undirected graph with a given degree sequence, it is always possible to use the Havel-Hakimi procedure to generate a given instance. Nevertheless, it is not necessarily the case for any target set.

3.5 Complexity and limitations of the method

In this section, we compute the theoretical complexity of the process. Then, we discuss the technical limitations that a user may encounter, which are not explicit through the complexity analysis. These limitations stem from the mixing problem of the Markov chain.

3.5.1 Complexity of the Markov process

We can express the time complexity of the *pks* process as a function of s, the number of steps (or k-swap trials), μ_k the expectation of the number of edges involved in a trial and structural features of the graph, typically n and m.

At each step, k edges are selected for swapping, and possibly actually swapped. An appropriate data structure allows to implement the selection and swap with an amortized complexity of $\mathcal{O}(k)$. That is the case of the implementation provided. In addition to this, we have to check if the target properties are satisfied by the modified graph. This depends on the nature of the properties, and must be computed for each specific case.

In the simplest case, we only check if the graph remains simple, i.e, we do not create loops or multi-edges, which can be done in $\mathcal{O}(k)$. Therefore, the amortized complexity of the method in this simple case is in $\mathcal{O}(\sum_1^s k.P(k)) = \mathcal{O}(s.\mu_k)$. The expression of μ_k only depends on the probabilistic law chosen to draw k.

3.5.2 Mixing time and convergence criterion

The main weakness of the MCMC sampling methods is the fact that there is no universal theoretical argument to know the number of steps needed to reach the stationary state *a priori*. Moreover, this question is known to be very difficult to address theoretically. Therefore, defining an experimental criterion to decide when the steady state is reached is the common practice. The experimenter usually chooses a "quantity of interest", *e.g.*, the assortativity of the graph, its number of triangles, its diameter, etc. and evaluates its convergence along the MCMC process [17]. In [47], the authors make the assumption based on experimental studies that there is a linear relation between the number of edges m of the graph and the number of necessary swaps to reach the steady state, at least in the case of connected graphs with a given degree sequence.

To address the problem in a robust and systematic way, experimental criteria have been designed to evaluate the convergence of MCMC sampling methods. For instance, Stanton and Pinar [41] investigate the integrated autocorrelation time of each edge and evaluate with an experimental threshold when they can consider that they have reached the steady state. It is also common practice to use time-series convergence diagnostics on quantities of interest, such as the Gelman-Rubin or the Raftery-Lewis tests [36]. Recently, Dutta *et al.* [13] showed that their criterion based on the Dickey-Fuller Generalized Least Squares test is more appropriate to sample graphs than other diagnostic methods. Note that while most of these techniques are applied in the context of graphs with a given degree sequence in various graph families (multigraphs or simple graphs, with or without self-loops), there is no fundamental obstacle to use them with other types of MCMC sampling methods.

While there is a range of possibilities, we propose here to apply a procedure close to the one proposed by Dutta *et al.* in [13] which has been designed specifically for the swap-based method described in [15], as it appears to be very comprehensive and safe.

3.5.3 Limitations

A user of the method should be aware that there is no guarantee in general of rapid mixing with MCMC sampling methods. Therefore, it is possible that a target set of constraints does not lead to a rapid mixing on a specific input, which translates to an unacceptably slow convergence.

In particular, when large k values are needed to reach ergodicity, we may experience a dramatic drop in the success rates of the Markov process. An illustrative toy example is detailed in [33]: the author considers a specific family of graphs with a given degree sequence and number of triangles such that $k = 8$ edge swaps are necessary to reach ergodicity with this set of constraints. It means in practice that any k-swap with $k < 8$

will fail, but also that 8 or more swaps are very unlikely to succeed because those leading to other elements of the set are only a small fraction among all combinations.

Fortunately, such configurations are rather unusual in real networks, and thus we assume that in many interesting practical cases, large k values are not needed. In any event, this question has to be investigated experimentally, which is what we develop in the next section.

4 Experiments

In this section, we implement experimentally the *pks* method with several types of target sets that have attracted interest in the literature for various problems related to complex networks analysis. More precisely, we first consider different flavors of simple graphs with a given degree sequence, then we address the generation of graphs with additional constraints: a fixed joint degree matrix for undirected graphs, and a fixed number of mutualistic dyads for directed graphs.

Note that in the case of simple graphs, it is not different to sample vertex-labeled graphs and stub-labeled graph, as discussed in details in [15]: *"the choice of graph labeling is inconsequential for studies of simple graphs"*, the reason being essentially that the number of stub-labeled graphs corresponding to the same vertex-labeled graph is exactly identical for any graph of the sets considered. So the sampling that we are doing here is usable for simple labeled graphs in general.

Throughout our experiments, we are concerned with the uniformity of the sampling achieved and compare to other existing methods, when such a method is available. We also report the computation times, investigate how the process scales, which parts are time-consuming and what is the impact of the parameters.

4.1 Experimental evaluation of the mixing times

As mentioned before, several works present methods to evaluate experimentally the convergence of a MCMC based process in the specific context of graph generation and we chose to follow the one proposed by Dutta *et al.* in [13] because of its comprehensiveness. Then, we discuss the question of how to modify this process and accelerate it.

4.1.1 General scheme

In [13], the authors propose a comprehensive method to generate graphs with a given degree sequence in a standardized manner. The statistic chosen to characterize the progression of the Markov process is the network degree assortativity, as it has the advantage of being fast to compute and update. The generation process is separated into two phases:

- First, there is the estimation of the *sampling gap* η for the process. It is itself decomposed in two sub-parts: the *burn-in*, which intends at reaching the stationary state of the Markov process by applying a large number of iterations (typically $1000m$), and after the burn-in, the evaluation of η itself. The sampling gap is the number of iterations of the Markov process separating two samples, when the chain has reached its stationary state. It is supposed to be above the threshold so that the samples can be considered as uncorrelated. For that purpose, the authors of [13] set η to a small value and test for autocorrelation at lag-1. If the independence hypothesis is rejected, η is increased by a constant amount and the process is iterated, otherwise it is considered that the threshold value is reached. The smallest value η above the threshold will be used for the following step.

- Second, there is the actual sample generation phase: they run the Markov chain and create a set of sample graphs using η during the corresponding window. The convergence of the process is checked by analyzing the distribution of the network statistic over this window. By comparing different convergence methods for this purpose, they come to the conclusion that the most appropriate one is the Dickey-Fuller Generalized Least Squares test (or DFGLS), as it ensures to obtain both a good accuracy and time-efficiency.

For the implementation details of the autocorrelation at lag-1 and DFGLS tests (level of significance, parameters, etc), we point the interested reader to the original paper [13] and underline that we can make similar choices as they do not depend on the nature of the target set.

We give in Algorithm 1 a summarized version of the generation part of the process as we implement it, without going into the details of the tests which depend on the target properties.

Algorithm 1 Summarized *pks* sample generation

Input: $G_0 = (V_0, E_0) \in$ Target Set; sampling gap η; sample size N; statistic f

$t = 0$; $T \leftarrow False$; $S \leftarrow \emptyset$
while not T **do**
  $G_{t+1} \leftarrow k$-SWAP(G_t)
  $t = t + 1$
  if $t \equiv 0$ $(mod\ N.\eta)$ **then**
    $S \leftarrow \{G_{t-N.\eta}, G_{t-(N-1).\eta}, \ldots, G_t\}$ ▷ Collecting sample
    $T \leftarrow$ DFGLS-test(f, S) ▷ Testing convergence
return S
procedure k-SWAP(G)
  draw k according to $P(k)$ distribution
  draw k edges $(u_1, v_1), \ldots, (u_k, v_k) \in E$
  draw permutation σ of $[\![1 : k]\!]$
  $E' \leftarrow E \setminus \{(u_1, v_1), \ldots, (u_k, v_k)\} \cup \{(u_1, v_{\sigma(1)}), \ldots, (u_k, v_{\sigma(k)})\}$
  $G' \leftarrow (V, E')$
  if $G' \in$ Target Set **then**
    return G'
  else
    return G

4.1.2 Accelerating the computation

It is difficult to reduce the time spent on the generation phase itself, which depends on the size of the sample to produce. By contrast, there is some room to reduce the computation time of the first phase (sampling gap estimation) of the process.

Dutta *et al.* [13] have already made this observation and proposed acceleration heuristics adapted to the case of graphs with a given degree sequence in various spaces. They observed experimentally that a typical threshold for η value is a few m, which is consistent with previous experimental observations from other authors [17, 47] and with the decision tree proposed in [13] depending on the graph density. Besides, it is intuitively reasonable that there is a linear relationship between m and the number of swaps that is necessary

to randomize the graph, as a swap is itself a modification of a set of edges. Unfortunately, these heuristics are not adapted to the case of various target sets that we investigate. Indeed, more constrained target sets imply that the success rate ρ of the Markov process drops and we intuitively expect that the number of edges should be related to the number of swap *successes* rather than the number of swap *attempts*. In short, any acceleration scheme that does not take ρ into account is certainly not suited to our study.

However, we reduce the computation time of the sampling gap. According to Algorithm 1 in [13], η initial value is chosen equal to 0 then it is increased by the same constant amount $0.05m$. Practically, it implies some useless tests of the autocorrelation at lag-1. Therefore, we propose to choose the initial η value by assessing ρ during the burn-in phase of the process and to start the search of the sampling gap with $\eta = m/\rho$. Then, instead of looking for η with a linear search, we use a binary search: if the stationary test is rejected, we change η to 2η and iterate the process, if it is not, we change η to $\eta/2$ and iterate the process. This recursive procedure is halted as soon as the behavior changes: if the stationary test was rejected and is now accepted, we consider that we have reached the threshold by lower values ; alternatively if the test was accepted and is now rejected, we consider that the penultimate η value was just above the threshold. Note that the gain between our binary search and the original sequential search is not significant when ρ is close to 1. But we will also see examples where ρ is much lower than 1 which leads to significantly faster binary searches than sequential searches.

4.2 Datasets

For comparison and reproducibility purposes, we use public datasets investigated in the literature, principally extracted from Konect database `http://konect.cc/`. We explore datasets of different sizes to evaluate how the procedure scales up. Depending on the cases considered (undirected or directed graphs, bipartite or not) we use different datasets for various sets of constraints. Note that all graphs considered are simple (no multi-edge, no loop) and that bipartite graphs are also undirected. We summarize in Table 1 some global structural properties of the datasets considered, as well as the sources used.

Dataset	*type*	*family*	n	m	*source*
Zachary	`m-u`	friendship	34	78	[24]
Les Misérables	`m-u`	co-appearance	77	254	[24]
Powergrid	`m-u`	infrastructure	4941	6594	[24]
Managers	`m-d`	friendship	21	100	[23]
Yeast	`m-d`	gene regulation	688	1079	[29]
Air Traffic	`m-d`	infrastructure	1226	2615	[24]
Finches	`b-u`	co-occurrence	17;19	55	[18]
Chiloe	`b-u`	mutualistic	129;26	312	[39]
Crime	`b-u`	person-crime	829;551	1476	[24]

Table 1: Global structural properties of the graph datasets investigated. The type depicts if the network is monopartite (`m`) or bipartite (`b`), undirected (`u`) or directed (`d`). We also indicate what is the network family; their number of nodes n and edges m and the source in the literature where this dataset can be found.

4.3 Implementation and hardware

All the experiments are run on a server with 252GB of RAM and 4 64-bit Intel Xeon E5-4617-0 CPUs with 6 cores at 2.90GHz. The implementations of the code are in python, the source code and documentation are available at the address `https://gitlab.lip6.fr/tabourier/code-pks-generation`. As described in the documentation, the different constraints are indicated as arguments of the program, so that there is a unique implementation for all the variants described in what follows. Note also that the η estimation time implementation is by default parallelized on 4 threads, if allowed by the system and hardware.

4.4 Fixed degree sequences

We implement several flavors of models with fixed degree sequences. Namely, we consider simple monopartite undirected graphs, simple monopartite directed graphs and simple bipartite undirected graphs, all three of them with fixed degree sequences. In all these cases, we only have to check if a k-swap generates a loop or a multi-edge, which can be done in $\mathcal{O}(k)$. It makes the complexity of the process in $\mathcal{O}(s.\mu_k)$ with the notations given in Section 3.5.

There are existing methods to generate such graphs, in particular the `Curveball` method by Carstens *et al.*, described in [9], which is also a MCMC method relying on a different elementary step. More precisely, we use the `global Curveball` method to generate uniform samples of such graphs, to which we compare the samples obtained with our method. It is available in `python NetworKit` at this address: `https://networkit.github.io/dev-docs/notebooks/Randomization.html`.

4.4.1 Simple monopartite undirected graphs

We first consider simple monopartite graphs. This class of graphs have been largely studied, as detailed in the state of the art Section. We investigate in more details the generation of the reference `m-u` graphs listed in Section 4.2.

In the default settings of our experiments, we generate a sample of 1000 graphs in each case, using the *pks* method, with $\gamma = 2$. The number of triangles of the graph is used as a statistic to follow the process, both for the autocorrelation at lag-1 test used to determine the sampling gap, and for the DFGLS test used to check the convergence of the process. We have chosen this quantity because it is fast to compute and update. Note that in [13], the authors used the assortativity measure, which is even faster to update, however the assortativity is a real value, which implies using bins in order to make statistical tests. For this reason, we favor the number of triangles.

In Table 2, we report the results of our experiments, which are the η obtained and the computation times of each part of the process: η estimation, convergence and generation of 1000 graphs. The computation times indicated are CPU times, but the wall clock times are actually shorter as the η estimation phase is parallelized (on 4 threads by default). We observe that the convergence time is negligible, and above all that the η estimation dominates the computation time, as it is typically 8 times larger than the sample generation time. Note that the sample generation time is a linear function of the number of graphs generated and can therefore be modulated depending on the user's needs.

To illustrate the convergence process, we follow the evolution of the distribution of triangles throughout the evaluation of the experimental η for the dataset *Les Misérables*. It is expected that its shape and average evolves before stabilizing when the stationary

Dataset	η	η estimation time (in s)	convergence time (in s)	sample gen. time (in s)	total CPU time (in s)
Zachary	751	1,731	2	221	1,954
Les Misérables	1,781	5,478	6	679	6,163
Powergrid	16,137	205,946	55	27,527	233,528

Table 2: Computation times for a 1000 graphs sample generation in the case of simple monopartite undirected graphs with a fixed degree sequence.

state of the process is reached. This evolution, reported in Figure 3, is consistent with this expectation.

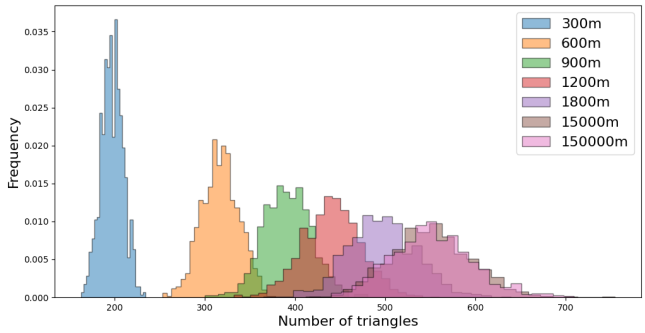

Figure 3: Evolution of the distribution of triangles through the convergence process on the dataset *Les Misérables*. Each distribution corresponds to a fixed value of the number of swap attempts (evaluated in m), for 1000 graphs. When the stationary state is reached, the distribution does not evolve significantly, as detected by the DFGLS test.

It is also possible to compare the distribution obtained in the stationary state of our process to the sample distribution obtained using the `global Curveball` method. Note that we set `global Curveball` *number of global rounds* parameter to 100 in order to guarantee the convergence of the process. To evaluate if these two distributions can be considered as produced from the same set, we use a two-sample Kolmogorov-Smirnov (KS) test on the number of triangle distribution of both samples. We remind that the KS-test evaluates if the samples can come from the same distribution, and a p-value larger than a given threshold (we choose a standard 0.05), indicates that this assumption cannot be rejected. With 1000 graph samples, we obtain p-values of 0.969 (*Zachary*), 0.501 (*Les Misérables*) and 0.723 (*Powergrid*), so largely above the fixed threshold.

For practical purposes, it is worth noticing that `global Curveball` is much faster than our method. A direct comparison is not simply doable because there is no equivalent to the automatic search for η in `global Curveball`. However, one can consider that the sample generation times in our *pks* method can be compared to `global Curveball` computation (CPU) times. With a number of global rounds of 100 and a sample of 1000 graphs, they are the following: 73s (*Zachary*), 75s (*Les Misérables*) 372s (*Powergrid*), to be compared

to the *pks* sample generation times in Table 2, so a ratio ranging from 3 to 78 times faster. Indeed, `global Curveball` is specifically designed for the purpose of generating graphs with a given degree sequence very efficiently, while the *pks* method favors versatility and adaptability at the cost of a lower efficiency.

4.4.2 Simple monopartite directed graphs

We consider the problem of generating a sample of simple directed graphs with a fixed degree sequence. This problem is different from the one examined in the previous section because of the edge directions, which makes the simple swap process reducible in the general case.

In Table 3, we report the generation times for our reference `m-d` graphs. We precise that here again, we use the number of triangles as a statistic. Although, several definitions of this concept are available in the context of directed graphs, we choose to use the number of triangles of the undirected graphs with the same set of nodes and edges but creating for each directed edge an undirected edge between the same nodes (possible multiple edges are converted to simple edges).

Dataset	η	η estimation time (in s)	convergence time (in s)	sample gen. time (in s)	total CPU time (in s)
Managers	1,762	1,563	21	313	1,897
Yeast	3,394	24,900	20	1,077	25,997
Air Traffic	6,719	33,905	35	3,877	37,817

Table 3: Computation times for a 1000 graphs sample generation in the case of simple monopartite directed graphs with a fixed degree sequence.

`global Curveball` can also produce uniform samples of directed graphs as long as a preprocessing step is enabled (see [6]). Thus we can also make a comparison between the 1000 graph samples obtained, using a Kolmogorov-Smirnov test on the distributions of the number of triangles of both samples. It yields p-values of 0.410 (*Managers*), 0.078 (*Yeast*) and 0.433 (*Air Traffic*), above the threshold demanded. Here also, `global Curveball` outperforms *pks* on the ground of speed, with computation times of 72s (*Managers*), 92s (*Yeast*) and 293s (*Air Traffic*), so 5 to 13 times faster than *pks* sample generation.

4.4.3 Simple bipartite undirected graphs

The third case of graphs with a fixed degree sequence that we examine are bipartite graphs with a given degree sequence. For bipartite graphs, it is not possible to follow the process with the number of triangles (which is 0 by definition), so we resort to the assortativity and report the results on our reference `b-u` graphs in Table 4.

Again, `global Curveball` is able to generate uniformly random bipartite graph with a given degree sequence, so we make a comparison between the samples obtained, using a Kolmogorov-Smirnov test on the assortativity distribution, as there is no triangle in a bipartite graph. It yields p-values of 0.055 (*Finches*), 0.648 (*Chiloe*) and 0.536 (*Crime*). These values are above the threshold, but only slightly concerning *Finches*. It is probably due to the fact that we are using a KS-test on a discrete distribution, which might generate Type I errors (unnecessary rejections), especially for highly discretized distributions which may happen with a small graph such as *Finches*. Concerning speed, `global Curveball`

Dataset	η	η estimation time (in s)	convergence time (in s)	generation time (in s)	total CPU time (in s)
Finches	930	993	1	111	1,005
Chiloe	3,042	11,067	21	539	11,627
Crime	3,869	12,416	16	1,453	13,885

Table 4: Computation times for a 1000 graphs sample generation in the case of simple bipartite undirected graphs with fixed degree sequences.

computation times are 73s (*Finches*), 72s (*Chiloe*) and 302s (*Crime*), so from 1.5 to 5 times faster than *pks* sample generation.

4.5 Fixed joint degree matrix

Now, we consider the generation of uniform samples of graphs with a more elaborate set of properties, namely simple undirected graphs with a fixed joint degree matrix. By definition, the element (j, k) of the joint degree matrix (JDM) contains the number of edges connecting nodes of degree j to nodes of degree k. Consequently, setting the JDM implies setting the degree sequence.

Note that *a priori* `global Curveball` is not usable in this context, as nothing ensures the irreducibility of the associated Markov Chain. However, the method proposed by Stanton and Pinar [41] allows to generate uniform samples of graphs with a fixed JDM. Thus, we will use it for comparison purposes. The related program, named `graphMC` is currently available online at `https://www.sandia.gov/-jairay/open-source-software/` with details in paper [36].

In Table 5, we report the generation times for the `m-u` graphs that we consider in this work. We follow the process with the number of triangles in the graph.

Dataset	η	η estimation time (in s)	convergence time (in s)	sample gen. time (in s)	total CPU time (in s)
Zachary	5,053	2,811	5	1,368	4,184
Les Misérables	24,643	17,824	28	8,546	26,398
Powergrid	197,180	1,682,480	6,023	313,048	2,001,551

Table 5: Computation times for a 1000 graphs sample generation in the case of simple monopartite undirected graphs with fixed degree sequences and fixed joint degree matrices.

In this case, we see that the η estimation phase dominates the total computation time, in a ratio which is roughly 2/3 to 1/3, when generating a 1000 graphs sample. By comparison to the case of simple monopartite graphs with a fixed degree sequence, we observe that the estimated η as well as the overall generation times are much larger. It is expected as the constraint is stronger, we know that the target set with a fixed JDM is strictly included in the one with a fixed degree sequence (discussed in Section 4.4.1).

We quantify this observation by measuring the success rates ρ of swaps in these two series of experiments. The results are reported in Table 6. We can see that on the graphs examined the success rates typically drop by a factor which is up to 27 on *Powergrid* dataset. This factor is simply indicative, as it varies significantly from a graph to another,

but it reflects the fact that much less graphs have the same JDM as the original graph. In particular, the success rate appears to be much smaller than 1, which illustrates the remark about the sampling gap computation: assessing the success rate ρ during the burn-in phase is useful as it can be low when the target set is heavily constrained.

	degree sequence only		joint degree matrix	
Dataset	η	ρ	η	ρ
Zachary	751	10.39%	5,053	1.54%
Les Misérables	1,781	14.26%	24,643	1.03%
Powergrid	16,137	40.86%	197,180	1.54%

Table 6: Comparison between the *pks* success rates for the generation processes of a set of graphs with fixed degree sequence only and a set of graphs with fixed joint degree matrix.

The samples are compared to the ones that are produced with `GraphMC`. To evaluate if these two distributions can be considered as produced from the same set, we use a Kolmogorov-Smirnov test with 1000 graph samples, which yield the following p-values: 0.573 (*Zachary*), 0.954 (*Les Misérables*) and 0.936 (*Powergrid*), above the fixed 0.05 threshold, which confirms that the samples obtained with `graphMC` and our method are statistically similar.

Similarly to the discussion in Section 4.4, we can compare the computation times of `graphMC` to *pks* sample generation times, although we expect that `graphMC` is more efficient because it is especially designed to generate graphs with a fixed joint degree matrix and because it is coded in C++ which is a more efficient language than python. The parameter `N` in `GraphMC` can be related to η by the following relation: $m.\texttt{N} = S.\eta$ for a sample of size S This is indeed the case, we observe the following CPU times: 15s (*Zachary*), 148s (*Les Misérables*) and 66,950s (*Powergrid*). We can see that `GraphMC` outperforms our method by a factor from 5 to 90 on these examples. More generally, it is fair to assume for practical usages that a specialist method should outperform *pks*, but the advantage of our method emerges when considering target sets where there is no other method available.

4.6 Fixed degree sequences and number of mutual dyads

Finally, we examine the problem of generating a set of directed graphs with a given degree sequence and a fixed number of mutual dyads. Mutual dyads are reciprocal connections between two nodes, and they have been especially studied in the context of social networks where mutual relationships have a completely different meaning from non-mutual relationships. In particular, in social graphs, reciprocal relationships are known to be over-represented when compared to simple graphs with a fixed degree sequence only. While several attempts have been made to generate uniformly random graphs obeying these constraints, the ones by Roberts [37] then McDonald *et al.* [26] failed to be actually uniform. Tao [43] proposed an algorithm for these graphs which is uniformly random; unfortunately, there is no implementation available. So, to the best of our knowledge, our method is currently the only one available to generate such graphs.

We report the results of the generation process in Table 7 on our reference `m-d` graphs. Here we can see that η estimation times largely dominate the overall generation times, as it ranges from 8 to 17 more time than the sample generation itself, for 1000 graph samples.

Dataset	η	η estimation time (in s)	convergence time (in s)	sample gen. time (in s)	total CPU time (in s)
Managers	4,892	15,377	88	879	16,344
Yeast	3,426	24,273	7	3,006	27,386
Air Traffic	12,332	585,711	264	24,797	610,772

Table 7: Computation times for a 1000 graphs sample generation in the case of simple monopartite directed graphs with a fixed degree sequence and a fixed total number of mutual dyads.

4.7 Impacts of γ and the graph statistic

We remind that the method has one parameter, which is the exponent γ when selecting the value of k at each step of the algorithm. A smaller γ induces a higher probability to select larger k. We know experimentally that this parameter is related to the convergence speed of the method, however the relation is not trivial as there are several counteracting effects. Indeed, a swap implying more edges has a higher probability to fail, but also, if the swap succeeds, it leads to a better mixing of the graph structure. It means that it is hard to predict what will be the effect of a larger γ on η and on the generation time. To have a better idea of the impact of this parameter, we make some experiments on the `m-u` graphs with a fixed degree sequence and test the following γ values: $\{2, 3, 4\}$.

We also investigate the impact of the choice of the network statistics on the estimation of the convergence. This statistic has two important roles throughout the protocol: for the autocorrelation at lag-1 which is used to estimate the sampling gap η, and for the DFGLS test which guarantees the convergence. This question has already been explored in other studies, in particular in [13, Appendix F], essentially concluding that all the statistics examined have converged when the convergence is detected according to the protocol with the assortativity. We compare the convergence estimations with the two statistics which are implemented in our code: the assortativity and the number of triangles on the `m-u` graphs with a fixed degree sequence.

▲	*Zachary*			*Les Misérables*			*Powergrid*		
γ	η	ρ	time (s)	η	ρ	time (s)	η	ρ	time (s)
2	751	10.39%	1,954	1,781	14.26%	6,163	16,137	40.86%	233,528
3	533	14.63%	1,075	1,281	19.83%	7,951	14,911	44.22%	204,742
4	456	17.09%	867	1,127	22.53%	2,496	14,264	46.23%	181,570
a	*Zachary*			*Les Misérables*			*Powergrid*		
γ	η	ρ	time (s)	η	ρ	time (s)	η	ρ	time (s)
2	743	10.53%	2,003	1,773	14.33%	5,954	16,125	40.89%	265,890
3	528	14.78%	976	1,290	19.70%	16,959	14,901	44.22%	206,511
4	457	17.08%	2,136	1,130	22.48%	5,883	14,265	46.23%	181,570

Table 8: Comparison of η (sampling gap), ρ (success rate) and total time in the case of undirected monopartite graphs with a fixed degree sequence for various γ, when the convergence is followed using the number of triangles (top) and the assortativity (bottom).

Results on both questions are summarized in Table 8. We know that lower k generally implies larger success rates ρ. We can decompose the success rates for different k values

and denote ρ_k the success rate corresponding to the value k of a k-swap. For instance, if we decompose ρ in the case of *Les Misérables* dataset, we observe that for $k = 2$ it is $\rho_2 \simeq 26.47\%$, while for $k = 3$ it is $\rho_3 \simeq 12.09\%$, for $k = 4$ it is $\rho_4 \simeq 9.98\%$, and ρ_k continues to drop as k increases. Consequently, as increasing γ implies lower k on average, we observe a higher success rates ρ. However, η and the generation times may vary differently: for *Les Misérables* dataset, we observe with $\gamma = 3$ that η is lower but the generation time is larger than with $\gamma = 2$. Here, we certainly see the effect of the fact that a lower γ induces a lower success rate but might result in a shorter convergence time because of a more efficient mixing.

Concerning the graph statistics (assortativity and number of triangles), we first note that η is roughly similar for both statistics, which is not surprising considering the fact that we are looking for η using a binary search. Second, ρ does not depend on the graph statistic, but only on the Markov process itself, therefore it is the same (except for statistical fluctuations) in both series of experiments. The most important point is the generation time and there is no definitive answer: in some cases, using the assortativity leads to shorter generation times than using the number of triangles, in others we see the opposite. In all cases, both statistics lead to the same order of magnitude of convergence time, which is consistent with the conclusion of [13].

Finally, we check the consistency of the samples obtained using KS tests, by choosing a reference sample ($\gamma = 2$, followed with the number of triangles) and comparing it to the other ones obtained for the same dataset. The p-values obtained are summarized in the Table 9 and show that in all cases they are above the 0.05 threshold, which shows that the different samples are consistent.

	$\gamma = 3$ (▲)	$\gamma = 4$ (▲)	$\gamma = 2$ (a)	$\gamma = 3$ (a)	$\gamma = 4$ (a)
Zachary	1.000	0.401	0.936	0.685	0.888
Les Misérables	0.573	0.370	0.994	0.969	0.314
Powergrid	1.000	0.980	0.610	0.914	0.536

Table 9: p-values of the KS-tests comparing the distribution of triangles of the samples obtained with different values of γ and graph statistics to the reference case ($\gamma = 2$, convergence followed with the number of triangles).

Conclusion

In this work, we have presented a methodology to generate uniform samples of simple graphs obeying a specific degree sequence and any additional property, provided that we have at least one element of the target set of graphs. It is based on a Monte Carlo Markov Chain method, and its principle is to select at each step a set of k edges, k being drawn from a predefined distribution, then to permute the extremities of these edges randomly. We also provide a documented implementation of this algorithm in python, which is designed to generate samples of undirected, directed and bipartite simple graphs with a given degree sequence, directed graphs with a given number of mutual dyads, or undirected graphs with a given joint degree matrix. We have shown experimentally that this generator is indeed able to produce random samples of graphs on these examples, in particular in the case of directed graphs with a given number of mutual dyads, for which no other method is currently available. The code has been designed to be modified by future users in order to adapt it to their own needs, by adjusting the set of constraints.

In that sense, this work opens the way to some interesting prospects. One of the first modification that comes to mind is to adapt it to generate samples of loopy- and multi-graphs, in the spirit of what has been done by Fosdick *et al.* in the context of the configuration model [15]. Another interesting lead is to integrate to the Markovian process the possibility to add or delete an edge, in the line of what has been proposed by Van Koevering *et al.* to generate random graphs with a given core sequence [45]. It would allow to generate uniform samples of a whole new variety of models. While this improvement sounds theoretically doable using the same probabilistic logic as *pks*, it also has a major shortcoming: MCMC sampling methods usually have no guarantee on the convergence time and practically, the convergence process can be very slow when considering specific input sequences and constraints. In particular, including the edge addition and deletion to the process allows to enlarge the target set so much that it might make the convergence times too long for practical purposes.

This brings back to the main practical issue encountered with MCMC sampling methods, which is the issue of the convergence time. We suggest that accelerating heuristics could be developed to reach convergence faster. Indeed, we believe that the burn-in phase may be eliminated, as we observe experimentally that m/ρ is generally a sufficient amount of steps to reach the stationary state, based on the autocorrelation test at lag-1. This remark calls for deeper experimental investigation.

Funding

This work was supported by the ANR (French National Agency of Research) through the ANR FiT LabCom.

Acknowledgements

We thank Fabrice Lécuyer, Esteban Bautista-Ruiz, as well as anonymous reviewers for their reading and relevant suggestions on the paper. We thank Jaideep Ray for his explanations on the `graphMC` method.

References

[1] Georgios Amanatidis, Bradley Green, and Milena Mihail. Connected realizations of joint-degree matrices. *Discrete Applied Mathematics*, 250:65–74, 2018.

[2] Andrii Arman, Pu Gao, and Nicholas Wormald. Fast uniform generation of random graphs with given degree sequences. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1371–1379. IEEE, 2019.

[3] Yael Artzy-Randrup and Lewi Stone. Generating uniformly distributed random networks. *Physical Review E*, 72(5):056708, 2005.

[4] Kevin E Bassler, Charo I Del Genio, Péter L Erdős, István Miklós, and Zoltán Toroczkai. Exact sampling of graphs with prescribed degree correlations. *New Journal of Physics*, 17(8):083052, 2015.

[5] Mohsen Bayati, Jeong Han Kim, and Amin Saberi. A sequential algorithm for generating random graphs. *Algorithmica*, 58(4):860–910, 2010.

[6] Annabell Berger and Corrie Jacobien Carstens. Smaller universes for uniform sampling of 0, 1-matrices with fixed row and column sums. *arXiv preprint arXiv:1803.02624*, 2018.

[7] Joseph Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet mathematics*, 6(4):489–522, 2011.

[8] Béla Bollobás. *Random graphs*. Number 73. Cambridge university press, 2001.

[9] Corrie Jacobien Carstens, Annabell Berger, and Giovanni Strona. Curveball: a new generation of sampling algorithms for graphs with fixed degree sequence. *arXiv preprint arXiv:1609.05137*, 2016.

[10] Éva Czabarka, Aaron Dutle, Péter L Erdős, and István Miklós. On realizations of a joint degree matrix. *Discrete Applied Mathematics*, 181:283–288, 2015.

[11] Gregorio D'Agostino, Antonio Scala, Vinko Zlatić, and Guido Caldarelli. Robustness and assortativity for diffusion-like processes in scale-free networks. *EPL (Europhysics Letters)*, 97(6):68006, 2012.

[12] Charo I Del Genio, Hyunju Kim, Zoltán Toroczkai, and Kevin E Bassler. Efficient and exact sampling of simple graphs with given arbitrary degree sequence. *PloS one*, 5(4), 2010.

[13] Upasana Dutta, Bailey K. Fosdick, and Aaron Clauset. Sampling random graphs with specified degree sequences. *arXiv preprint arXiv:2105.12120*, 2022.

[14] Péter L Erdös, Catherine Greenhill, Tamás Róbert Mezei, István Miklós, Dániel Soltész, and Lajos Soukup. The mixing time of the swap (switch) markov chains: a unified approach. *arXiv preprint arXiv:1903.06600*, 2019.

[15] Bailey K Fosdick, Daniel B Larremore, Joel Nishimura, and Johan Ugander. Configuring random graph models with fixed degree sequences. *SIAM Review*, 60(2):315–355, 2018.

[16] Minas Gjoka, Bálint Tillman, and Athina Markopoulou. Construction of simple graphs with a target joint degree matrix and beyond. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1553–1561. IEEE, 2015.

[17] C. Gkantsidis, M. Mihail, and E.W. Zegura. The markov chain simulation method for generating connected power law random graphs. In *Proc. 5th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2003.

[18] Nicholas J Gotelli. Null model analysis of species co-occurrence patterns. *Ecology*, 81(9):2606–2621, 2000.

[19] Paul W Holland and Samuel Leinhardt. An exponential family of probability distributions for directed graphs. *Journal of the american Statistical association*, 76(373):33–50, 1981.

[20] Mark Jerrum and Alistair Sinclair. Conductance and the rapid mixing property for markov chains: the approximation of permanent resolved. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 235–244, 1988.

[21] Mark Jerrum and Alistair Sinclair. The markov chain monte carlo method: an approach to approximate counting and integration. *Approximation Algorithms for NP-hard problems, PWS Publishing*, 1996.

[22] Brian Karrer and Mark EJ Newman. Random graphs containing arbitrary distributions of subgraphs. *Physical Review E*, 82(6):066118, 2010.

[23] David Krackhardt. Cognitive social structures. *Social networks*, 9(2):109–134, 1987.

[24] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*, pages 1343–1350, 2013.

[25] Priya Mahadevan, Dmitri Krioukov, Kevin Fall, and Amin Vahdat. Systematic topology analysis and generation using degree correlations. *ACM SIGCOMM Computer Communication Review*, 36(4):135–146, 2006.

[26] John W McDonald, Peter WF Smith, and Jonathan J Forster. Markov chain monte carlo exact inference for social networks. *Social Networks*, 29(1):127–136, 2007.

[27] István Miklós and János Podani. Randomization of presence–absence matrices: comments and new algorithms. *Ecology*, 85(1):86–92, 2004.

[28] Ron Milo, Nadav Kashtan, Shalev Itzkovitz, Mark EJ Newman, and Uri Alon. On the uniform generation of random graphs with prescribed degree sequences. *arXiv preprint cond-mat/0312028*, 2003.

[29] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[30] Mark EJ Newman. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002.

[31] Mark EJ Newman. Random graphs with clustering. *Physical review letters*, 103(5):058701, 2009.

[32] Mark EJ Newman. The configuration model. In *Networks*. Oxford university press, 2018.

[33] Joel Nishimura. Swap connectivity for two graph spaces between simple and pseudo graphs and disconnectivity for triangle constraints. *arXiv preprint arXiv:1704.01951*, 2017.

[34] Chiara Orsini, Marija M Dankulov, Pol Colomer-de Simón, Almerima Jamakovic, Priya Mahadevan, Amin Vahdat, Kevin E Bassler, Zoltán Toroczkai, Marián Boguná, Guido Caldarelli, et al. Quantifying randomness in real networks. *Nature communications*, 6(1):1–10, 2015.

[35] A Ramachandra Rao, Rabindranath Jana, and Suraj Bandyopadhyay. A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 225–242, 1996.

[36] Jaideep Ray, Ali Pinar, and C Seshadhri. A stopping criterion for markov chains when generating independent random graphs. *Journal of Complex Networks*, 3(2):204–220, 2015.

[37] John M Roberts Jr. Simple methods for simulating sociomatrices with given marginal totals. *Social Networks*, 22(3):273–283, 2000.

[38] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.

[39] Cecilia Smith-Ramírez, P Martinez, M Nunez, C González, and Juan J Armesto. Diversity, flower visitation frequency and generalism of pollinators in temperate rain forests of chiloé island, chile. *Botanical Journal of the Linnean Society*, 147(4):399–416, 2005.

[40] Tom AB Snijders. Statistical models for social networks. *Annual review of sociology*, 37:131–153, 2011.

[41] Isabelle Stanton and Ali Pinar. Constructing and sampling graphs with a prescribed joint degree distribution. *Journal of Experimental Algorithmics (JEA)*, 17:3–1, 2012.

[42] Lionel Tabourier, Camille Roth, and Jean-Philippe Cointet. Generating constrained random graphs using multiple edge switches. *Journal of Experimental Algorithmics (JEA)*, 16:1–1, 2011.

[43] Trevor Tao. An improved mcmc algorithm for generating random graphs from constrained distributions. *Network Science*, 4(1):117–139, 2016.

[44] Richard Taylor. Switchings constrained to 2-connectivity in simple graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(1):114–121, 1982.

[45] Katherine Van Koevering, Austin Benson, and Jon Kleinberg. Random graphs with prescribed k-core sequences: A new null model for network analysis. In *Proceedings of the Web Conference 2021*, pages 367–378, 2021.

[46] Norman D Verhelst. An efficient mcmc algorithm to sample binary matrices with fixed marginals. *Psychometrika*, 73(4):705, 2008.

[47] Fabien Viger and Matthieu Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. In *International Computing and Combinatorics Conference*, pages 440–449. Springer, 2005.

[48] Harrison C White, Scott A Boorman, and Ronald L Breiger. Social structure from multiple networks. i. blockmodels of roles and positions. *American journal of sociology*, 81(4):730–780, 1976.