Efficient and simple generation of random simple connected graphs with prescribed degree sequence

Fabien Viger[1,2], Matthieu Latapy[2]

{fabien,latapy}@liafa.jussieu.fr

Abstract

We address here the problem of generating random graphs uniformly from the set of simple connected graphs having a prescribed degree sequence. Our goal is to provide an algorithm designed for practical use both because of its ability to generate very large graphs (efficiency) and because it is easy to implement (simplicity).

We focus on a family of heuristics for which we introduce optimality conditions, and show how this optimality can be reached in practice. We then propose a different approach, specifically designed for real-world degree distributions, which outperforms the first one. Based on a conjecture which we argue rigorously and which was confirmed by strong empirical evidence, we finally reduce the best asymptotical complexity bond known so far.

1 Introduction

In the context of large complex networks, the generation of random[3] graphs is intensively used for simulations of various kinds. Until recently, the main model was the Erdös and Renyi [7, 3] one. Many recent studies however gave evidence of the fact that most real-world networks have several properties in common [19, 2, 5, 20] which make them very different from random graphs. Among those, it appeared that the degree distribution of most real-world complex networks is well approximated by a power law, and that this unexpected feature has a crucial impact on many phenomena of interest [4, 20, 19, 8]. Since then, many models have been introduced to capture this feature. In particular, the Molloy and Reed model [17], on which we will focus, generates a random graph with prescribed degree sequence in linear time. However, this model produces graphs that are neither *simple*[4] nor *connected*. To bypass this problem, one generally simply removes multiple edges and loops, and then keeps only the largest connected component. Apart from the expected size of this component [18, 1], very little is known about the impact of these removals on the obtained graphs, which is usually either neglected or even forgotten. However, we gained confidence through numerous experiments that, in most cases corresponding to real-world heterogenous degree distributions, this process induces a significant skew on the degree sequence itself. A quantitative insight into the consequences of such biases is provided in the appendix.

The problem we address here is the following: given a degree sequence, we want to generate a random simple connected graph having *exactly* this degree sequence. Moreover, we want to be able to generate very large such graphs, typically with more than one million vertices, as often needed in simulations.

Although it has been widely investigated, it is still an open problem to directly generate such a random graph, or even to enumerate them in polynomial time, even without the connectivity requirement [21, 15, 16].

In this paper, we will first present the best solution proposed so far [9, 16], discussing both theoretical and practical considerations. We will then deepen the study of this algorithm, which will lead us to an improvement that makes it optimal among its family. Furthermore, we will propose a new approach solving the problem at much lower cost, and being very simple to implement.

[1]LIP6, CNRS and University Pierre and Marie Curie, 4 place Jussieu, 75005 Paris

[2]LIAFA, CNRS and University Denis Diderot, 2 place Jussieu, 75005 Paris

[3] In all the paper and unless otherwise specified, *random* means *uniformly at random*: each graph in the considered class is sampled with the same probability.

[4] A simple graph has neither multiple edges, *i.e.* several edges binding the same pair of vertices, nor loops, *i.e.* edges binding a vertex to itself.

Preliminary

Conventions and Notations

Throughout this paper, we use the following conventions :

- Graphs are undirected. n denotes the number of vertices of a graph, and m denotes the number of its edges.
- If a and b are vertices, $(a-b)$ denotes the edge binding them.
- Given four vertices a, b, c, d such that both edges $(a-b)$ and $(c-d)$ exist, the *edge swap* $(a-b), (c-d) \to (a-d), (b-c)$ consists in replacing these edges by the edges $(a-d)$ and $(b-c)$ (see Figure 1).
- Considering an edge swap $(a-b), (c-d) \to (a-d), (b-c)$, its *dual* edge swap is $(b-a), (c-d) \to (b-d), (a-c)$ (see also Figure 1).
- Considering an edge swap $(a-b), (c-d) \to (a-d), (b-c)$, its *reverse* edge swap is $(b-c), (d-a) \to (b-a), (c-d)$. Applying one edge swap and its reverse subsequently leaves the graph unchanged.

Complexity requirements and implementation

We assume the implicit use of graphs implemented so that the following complexities are ensured:

1. The existence of an edge between any pair (a, b) of vertices must be determined in $O(1)$ average time.
2. Given any four vertices a, b, c, d, the edge swap (see Fig. 1) $(a-b), (c-d) \to (a-d), (b-c)$ must be done in $O(1)$ average time.
3. For any vertex v, obtaining the set of the neighbors of v must be done in $O(d(v))$ time, where $d(v)$ is the number of such neighbors.
4. Picking a random edge (*i.e.* a pair (a,b) of vertices selected uniformly at random among all pairs of linked vertices) must be done in $O(1)$ average time.
5. The memory space required by the graph structure is linear in $n + m$.

Figure 1: An edge swap and its dual

A sketch of such an implementation[5] is described here. First, the vertices are identified by integers from 0 to $n-1$, and their degrees are stored in a separate array $Deg[\cdot]$. We assume that the architecture we use needs $O(1)$ time for elementary arithmetic operations on integers. For each vertex i, an adjacency list L_i is maintained, which is implemented as a hash table containing the $Deg[i]$ neighbors of i. Its address and memory space are respectively stored as $Addr[i]$ and $Size[i]$ in the separate arrays $Addr[\cdot]$ and $Size[\cdot]$. Natural hash table properties then ensure that the conditions 1 and 2 above are satisfied. We also make sure that for some fixed k, all hash tables are K-linear, *i.e.* $\forall i, Deg[i] \leq Size[i] \leq K \cdot Deg[i]$, so that point 3 is simply achieved by scanning the full hash table L_v, and removing the $O((K-1) \cdot Deg[i])$ gaps.

Point 4 is more delicate. To pick a random edge, we proceed in two steps: first, we pick a random vertex v so that the probability for each vertex to be elected is proportional to its degree. This can be done in $O(1)$ time if we list each vertex i exactly $Deg[i]$ times in a giant array $Prob[i]$ of total size $2m$. Then, we pick a random element w uniformly among all elements of L_v, which can be done in

[5]The implementation we actually used and made available online is slightly different, as it was further optimized for both time and memory issues. For more information, see the source code available at [25]

$O(K) = O(1)$ average time thanks to the K-linearity of the hash tables. At this point, the edge v, w has been picked uniformly at random among all edges.

Notice that all the structures that are needed to reach such complexities, *i.e.* the arrays $Deg[\cdot]$, $Prob[\cdot]$, $Addr[\cdot]$ and $Size[\cdot]$ aren't modified by any of the operations listed above. In particular, an edge swap doesn't change the degrees of the concerned vertices, and thus doesn't change the size of the concerned adjacency lists. They also meet the linear space requirement cited in point 5.

2 Context

2.1 The Markov chain Monte-Carlo algorithm

Several techniques have been proposed to solve the problem we address. We will focus here on the Markov chain Monte-Carlo algorithm [9], pointed out recently by an extensive study [16] as the most efficient one.
The generation process is composed of three main steps:

1. **Realize the sequence**: generate a simple graph that matches the degree sequence,
2. **Connect** this graph, without changing its degrees, and
3. **Shuffle** the edges to make it random, while keeping it connected and simple.

The Havel-Hakimi algorithm [11, 10] solves the first step in linear time and space. A result of Erdös and Gallai [6] shows that this algorithm succeeds if the degree sequence is realizable, *i.e.* if there exists a simple graph matching this degree sequence.

The second step starts by checking that $m \geq n-1$ and also that no vertex has degree zero, since the graph cannot be connected otherwise. Then, we use edge swaps to merge all the connected components into a single connected component. More precisely, given any two distinct connected components C_x and C_y such that C_x is not a tree, we pick an edge $(a-b)$ from C_x that can be removed without disconnecting C_x, and we also pick any edge $(c-d)$ from C_y. Now, the edge swap $(a-b),(c-d) \rightarrow (a-d),(b-c)$ clearly merges C_x and C_y into a single connected component, and does not create multiple edges since C_x and C_y were not connected to each other. See Figure 2 for an example of such operation. Thus, we can repeat this operation until the only components that are left are either a single connected component, or several trees, the latter being impossible as it would imply $m < n-1$. The linear complexity of this algorithm is straightforward.

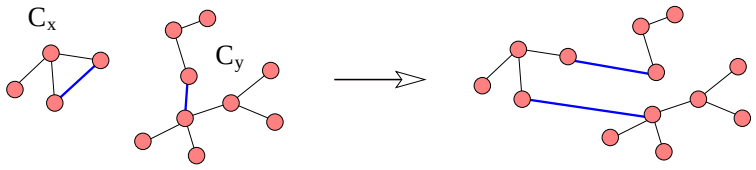

Figure 2: Using an edge swap to merging two separated components

The third step is achieved through random edge swaps. Given the graph G_t at some step t, we pick two edges uniformly at random and we swap them,obtaining another graph G' with the same degrees. If G' is simple and connected, we consider the swap as *valid*: $G_{t+1} = G'$. Otherwise, we reject the swap: $G_{t+1} = G_t$

This algorithm is a Markov chain where the **space** S_{CS} is the set of all simple connected graphs with the given degree sequence, the **initial state** G_0 is the graph obtained by the first two steps, and the **transition** $G_x \rightarrow G_y$ has probability $1/m(m-1)$ if there exists an edge swap that transforms G_x in G_y. If there are no such swap, this transition has probability 0. The latter comes from the two following observations:

1. If there exist at least one valid edge swap transforming G_x into G_y, this edge swap is *unique*

2. There are exactly $m \cdot (m-1)$ possible edge swaps (valid or invalid), since there are $m(m-1)/2$ possible unordered pair of edges, each unordered pair corresponding to exactly two edge swaps (see Fig. 1).

Moreover, for any graph $G_x \in S_{CS}$, the probability of the *self-transition* $G_x \to G_x$ is given by the number of invalid edge swaps on G_x, divided by $m(m-1)$. Now, let us assume in the rest of this paper that $n \geq 3$. Thus, G_x must contain at least one subgraph of the form a, b, c where both edges $(a-b)$ and $(b-c)$ exist. It is easy to see that the edge swap $(a, b, b, c-i), (s-i) \to (a, b, b, c-i), (i-s)$ nvalid, since it creates the edge $(b-b)$. Therefore, for every $G_x \in S_{CS}$, the probability $P_{G_x \to G_x}$ of the self-transition is greater than 0. Actually, $P_{G_x \to G_x} > 1/m(m-1)$.

Theorem 1. *This Markov chain is irreducible [23], symmetric, and aperiodic.*

Corollary 2. *The Markov chain converges to the uniform distribution on every states of its space,* i.e. *all graphs having the wanted properties.*

Proof. The corollary is a well-known consequence of the standard Markov chain theory.

- The irreducibility, *i.e.* the reachability of any state $G_y \in$ *spacecs* from any other state $G_x \in S_{CS}$, has been proven by Taylor [23].
- The aperiodicity comes from the fact that for any state $G_x \in S_{CS}$, the self-transition $G - x \to G_x$ has a striclty positive probability. Standard markov chain theory then ensures the aperidodicity.
- The symmetry comes from the reversibility of any edge swap: as discussed in the preliminary, if the edge swap $(a-b), (c-d) \to (a-d), (b-c)$ changes G_x into G_y, then the reverse edge swap $(a-d), (c-b) \to (a-b), (d-c)$ changes G_y into G_x. Therefore, $\forall G_x, G_y \in S_{CS}, P_{G_x \to G_y} = P_{G_y \to G_x}$

□

These results show that, in order to generate a random graph, it is sufficient to do *enough* transitions. However, no formal result has yet proved the existence of polynomial bounds in $n + m$ for the number of transitions needed to reach the convergence, even if recent work [14] tend to indicate that a quadratic bound might be established in the future. Since our contributions aims at reducing the average cost of the shuffle, regardless of its duration in terms of number of required edge swaps, we left further discussion about the convergence speed to future work, and used the following convention to evaluate the cost of our algorithms:

Convention 1 (Algorithmic Cost)**.** *The* ***cost*** *of our shuffle algorithms is defined as the average cost per edge swap. Edge swaps being cancelled during the algorithm are not counted. In the rest of the paper, we use the terms* ***unitary cost*** *to avoid misunderstanding.*

Nevertheless, to be able to display complexity measures concerning the global algorithm, we assumed the following, based on thorough empirical studies and on our own experience:

Convention 2. *[16, 9] The Markov chain converges after $O(m)$ edge swaps.*

This result was found in massive experiments [9, 16] where, even when using extremely biased initial graphs, $O(m)$ edge swaps were always sufficient to make the graph appear to be "really" random. More precisely, the distributions of a large set of non-trivial metrics (such as the diameter, the flow, and so on) over the sampled graphs were not different from the distributions obtained with random graphs. Notice that we tried, unsuccessfully, to find a metric that would prove this assertion false. Notice also that $\Omega(m)$ is a trivial lower bound for the convergence of the markov chain, since every transition only involves 2 edges among a total of m edges.

2.2 Equivalence between swaps and transitions

According to our definitions, a transition doesn't always imply an edge swap: whenever the edge swap attempt leads to an invalid graph, the transition has no effect, despite its computationnal cost. In the past litterature, the ratio of invalid edge swaps, though unknown, has either been considered as neglectable or simply forgotten. We provide here a short proof that this ratio is bounded in most cases, and discuss the implications of such a fact. Let us consider a simple connected graph G, and two edges

$(a-b),(c-d) \in G$, as shown in Figure 1. Now, let us remove these two edges to obtain G^*. Since G was connected, it is easy to see that either a or b is still connected to c or d in G^*. Now, if the edge swap $(a-b),(c-d) \rightarrow (a-d),(b-c)$ disconnects G, then in G^* it is clear that a is not connected to c, neither is b to d. Therefore, either a and d are connected or b and c are connected (in G^*), so that the dual edge swap $(b-a),(c-d) \rightarrow (b-d),(a-c)$ does not disconnect G. Moreover, it doesn't create multiple edges, since a was not connected to c, nor b was to d. Therefore, it is straightforward that at most 50% of the edge swaps disconnect the graph.

Unfortunately, the existence of worst-case graphs where any edge swap creates loops or mutiple edges (the star graphs, the clique graphs, $\cdots$) makes it impossible to bound the ratio of swaps that create multiple edges. In order to assert such a bound, one has to assume some additional properties on G. Among numerous solutions, we finally settled to the following, which seemed the most natural for the class of graphs we work on.

Theorem 3. *For any simple connected graph, let us denote by ρ the fraction of all possible pairs of vertices which have distance greater than or equal to 3. The probability that a random edge swap is **valid** is at least $\frac{\rho}{2z(z+1)}$, where z is the average degree.*

Proof. If $\rho = 0$ the result is trivial. If $\rho > 0$, consider a pair (v,w) of vertices having distance $d(v,w) \geq 3$ (*i.e.* there exist no path of length lower than 3 between v and w). Since the graph is connected, there exists a path of length $l \geq 3$ $(v, v_1, \cdots, v_{l-1}, w)$ connecting v and w. The edge swap $(v,v_1)(w,v_{l-1}) \rightarrow (v,v_{l-1})(w,v_1)$ is **valid**: it does not disconnect the graph, and since the edges it creates could not pre-exist (else we would have $d(v,w) \leq 2$), it keeps it simple.

Now, the $\rho \cdot n(n-1)$ ordered pairs of vertices define at least $\frac{\rho \cdot n(n-1)}{8}$ edges swaps, since an edge swap corresponds to at most 8 ordered pairs. Therefore, a random edge swap is valid with probability at least $\frac{\rho \cdot n(n-1)}{8m(m-1)}$. The fact that $m = \frac{n \cdot z}{2}$ ends the proof. □

In practice, $\rho > 0$ (the connected graphs such that $\rho = 0$ are very particular, somewhere between a clique and a star), and its value tends to grow with the size of the graph, for a fixed degree distribution.

This will allow us, in the rest of the paper, to assert that for a given degree sequence, the ratio of valid edge swaps is greater or equal than some positive constant, which is independent of the graph. In particular, this result allows us to rephrase Convention 1 for the naive algorithm: the cost of our algorithms is proportionnal to the average cost per transition.

2.3 Complexity

As we have already seen, the first two steps of the random generation (realization of the degree sequence and connection of the graph) are done in $O(m)$ time. We saw that the last step requires at least $O\Omega(m)$ transitions. We will therefore naturally focus on the *unitary cost* of the shuffle, *i.e.* the average cost of the shuffle per validated edge swap. The discussion presented in Section 2.2 ensure that this definition is equivalent to the average cost per transition. Now, in a first naive implementation of the shuffle algorithm, each of them consists in an edge swap, a simplicity test, a connectivity test, and possibly the cancellation of the swap (*i.e.* one more edge swap). Using the graph implementation described in the preliminaries, the cost of an edge swap, a simplicity test, and a connectivity test are respectively $C_{swaps} = O(1)$, $C_{simp} = O(1)$ and $C_{conn} = O(m)$, which leads to a linear unitary cost:

$$C_{naive} = O(m)$$

One can however improve significantly this time complexity using the structures described in [12, 13, 24] to maintain connectivity in dynamic graphs. Each connectivity test can be performed in time $O(\log n / \log\log\log n)$ and each simplicity test in $O(\log n)$ time. An edge swap then has cost $O(\log n (\log\log n)^3)$. Thus, the unitary cost per transition becomes:

$$C_{dynamic} = O\left(\log n (\log\log n)^3\right) \tag{1}$$

Notice however that these structures are quite intricate, and that the constants are large for both time and space complexities (the latter being still linear, though). The naive algorithm, despite the fact

that it runs in $O(m)$ time per transition, is therefore generally used in practice since it has the advantage of being extremely easy to implement. Our contribution in this paper will be to show how it can be significantly improved while keeping it very simple, and that it can even outperform the dynamical algorithm.

2.4 Speed-up and the Gkantsidis et al. heuristics

Gkantsidis et al. proposed a simple way to speed-up the shuffle process [9] in the case of the naive implementation: instead of running a connectivity test for each transition, they do it every T transitions, for an integer T called the *speed-up window*. If the connectivity test fails, T transitions have to be cancelled, which will considerably slow down the shuffle process, but in case of a success, $T-1$ connectivity tests will have been spared. The shuffle process can no longer be considered as a simple Markov chain; however, it has been proved [22, 9] that Corollary 2 still holds, *i.e.* that this process converges to the uniform distribution, although it is now composed of a concatenation of Markov chains [9], and even if the graph may actually get disconnected and connected again between two connectivity test.

Thus, the unitary cost of connectivity tests is reduced by a factor T, but at the same time the swaps are more likely to get cancelled: with T swaps in a row, the graph has more chances to get disconnected than with a single one. Thus, we have to distinguish between *post-validated* edge swaps – that represent a real step forward in the shuffle process – and the edge swaps that get cancelled because the subsequent connectivity test fails. In other words, the equivalence between transitions and validated edge swaps is no longer valid.

Let us remind that the terms *unitary cost* designate the average cost *per post-validated edge swap*. Thus, the unitary cost of some operation Q still represents the actual cost in regards to the global shuffle process. We introduce the following quantity:

Definition 1 (Success ratio)**.** *The success ratio $r_i = r(T_i)$ at a given step is the probability that the graph obtained from G_i after the T_i edge swap attempts is still connected.*

Now, let us consider the unitary cost of all operations: the speed-up divides the cost of the connectivity tests by T and leaves the cost of the simplicity tests and the edge swaps unchanged, but in the end the unitary cost is multiplicated by $1/r(T)$ because of the lower succes ratio.

$$C_{Gkan}(T) = \frac{C_{attempt} + \dfrac{C_{conn}}{T}}{r(T)} = a\frac{b + \dfrac{m}{T}}{r(T)} \tag{2}$$

where $C_{attempt}$ represents the average cost of one edge swap attempt with its simplicity test and the optional cancellation of the swap: $C_{simp} + C_{swaps} \leq C_{attempt} \leq C_{simp} + 2C_{swaps}$, and where $a = \frac{C_{conn}}{m}$ and $b = m\frac{C_{attempt}}{C_{conn}}$ are constants depending on the implementation.

For $T = 1$, we obtain the unitary cost of the naive algorithm:

$$C_{Gkan}(1) = O\left((1+m)\cdot r(T)^{-1}\right) = O\left(\frac{1+m}{1-p}\right) = O\left(m\right) = C_{naive}$$

Defining a good value for T is not easy, since the behavior of the success ratio $r(T)$ is not known. Intuitively, if T is too large, the graph will get disconnected too often, and $r(T)$ will be too small. If on the contrary T is too small, then $r(T)$ will be large but the complexity improvement is reduced. To bypass this problem, Gkantsidis et al. used the following heuristics (see Figure 3).

Heuristics 1 (Gkantsidis et al. heuristics)**.** `IF` *the graph got disconnected after T swaps*
`THEN` $T \leftarrow T/2$
`ELSE` $T \leftarrow T+1$

They expect T to automatically adjust itself so that it reaches a compromise between a large window T and good success ratio $r(T)$. We proved clearly (see the Appendix) that the fact that T may change during the shuffle still doesn't harm the uniformity of the convergence, even though the variations of T depend on the shuffle process itself.

Because the window T dynamically varies along the shuffle process, we had to arrange our notations. We divide the shuffle process in *steps*:

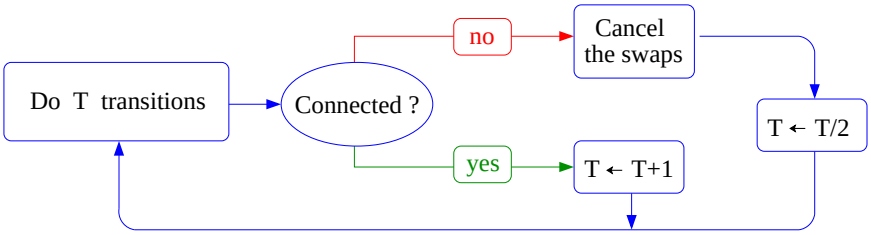

Figure 3: Heuristics 1 (Gkantsidis et al. heuristics)

Definition 2 (Steps of the shuffle process)**.** *In the shuffle process of Gkantsidis et al., a step is composed of the following operations:*

1. *Make a backup copy of the graph*
2. *Perform T edge swap attemps (each of them consists in one edge swap, one simplicity test and the reverse edge swap if the test failed)*
3. *Test the connectivity of the new graph, and restore the graph to its original state if the test fails*
4. *Modify T according to the heuristics specification*

We will use the notation G_i to designate the graph at the beginning of step i. Thus, G_0 is the graph obtained right before the shuffle process starts. Similarly, we will respectively use T_i, $C_i = C_{Gkan}(T_i)$ and $r_i = r(T_i)$ instead of T, $C_{Gkan}(T)$ and $r(T)$, and so on for any quantity related to the graph.

3 More from the Gkantsidis et al. heuristics

The problem we address now is to estimate the efficiency of the Gkantsidis heuristics. First, we introduce a framework to evaluate the ideal value for the window T. Then, we analyze the behavior of the Gkantsidis et al. heuristics, and get an estimation of the difference between the speed-up factor they obtain and the optimal speed-up factor. We finally propose an improvement of this heuristics which reaches the optimal. We also give experimental evidences for the obtained performance.

3.1 The optimal window problem

We introduce the following quantity:

Definition 3 (Disconnection probability)**.** *Given a graph G, the* disconnection probability *p is the probability that the graph gets disconnected after a random edge swap.*

Now, let us assume the two following hypothesis:

Hypothesis 1. *At any step i, the disconnection probability p remains constant for at least the duration of the step,* i.e. *during the T consecutive transitions. This probability is designated by p_i.*

Hypothesis 2. *The probability that a disconnected graph gets reconnected with a random swap, called the* reconnection probability*, is equal to zero.*

Notice that these hypothesis are not true in general. They are however reasonable approximations in our context and will actually be confirmed in the following. The first hypothesis is discussed more thoroughly in Appendix B. Moreover, the second hypothesis is just a worst-case scenario, and we introduced it only to simplify the computations. Empirically, we found it to be almost true (the reconnection probability was lower, or of the order of $1/m$) for all our scale-free networks topologies.

Assuming these hypothesis, the success ratio r_i, which is the probability that the G_i stays connected after T_i swaps, is given by:

$$r_i = (1-p)_i^T \tag{3}$$

Now, let us eliminate a first trivial case where p is so small that C_{Gkan} can be as low as $O(C_{attempt})$: if p is inferior or of the order of $\frac{1}{m}$, one can simply set $T = m$ and still obtain a good success ratio

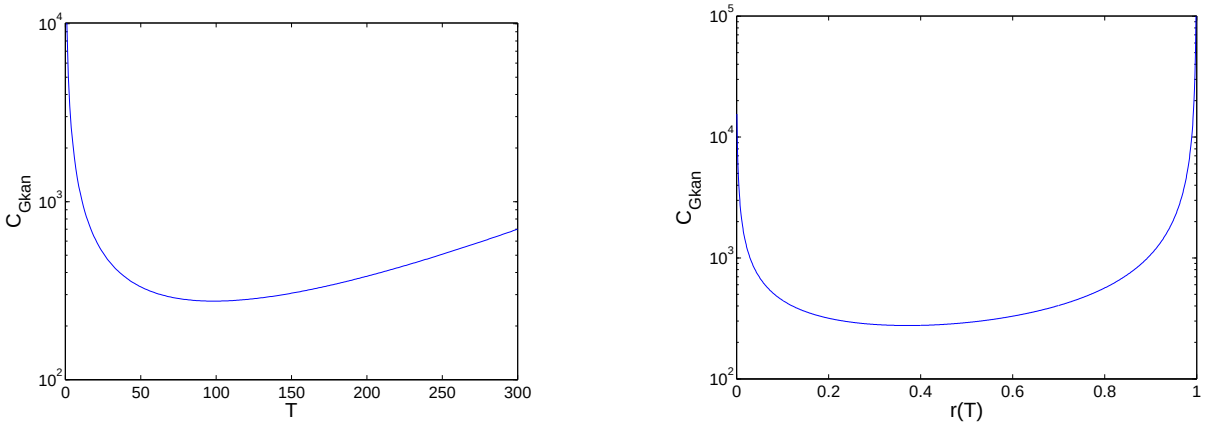

Figure 4: Evolution of the theoretical $C_{Gkan}(T)$ as a function of either T (left) or $r(T)$ (right). We used $a = b = 1$, $m = 10^4$ and $p = 10^{-2}$.

$r(T) = (1-p)^m = \Omega(1)$. Thus, Eq. 2 gives us $C_{Gkan} = O(1)$, which is asymptotically optimal. In the following, and for more simplicity, we will therefore eliminate this trivial case and assume that $p >> \frac{1}{m}$. This will make the formulaes much more understandable, without harming the nature of our arguments.

The optimal window T_i^{opt} at step i is the one that minimizes $C_{Gkan}(T_i)$. Using Eq.2 and after a bit of algebra, we obtain:

$$T_i^{opt} \cdot \left(\frac{b \cdot T_i^{opt}}{m} + 1 \right) = -log(1-p_i)^{-1}$$

which immediatly implies $T_i^{opt} < \frac{1}{p_i}$, and therefore $T_i^{opt} << m$ and $\frac{bT_i^{opt}}{m} + 1 \approx 1$ from the discussion stated above. This leads to:

$$T_i^{opt} = -log(1-p_i)^{-1} \tag{4}$$

Since the window T and the success ratio $r(T)$ are bijectively related, the optimality condition may also be considered as depending on r_i, which gives us an *optimal success ratio* at step i:

$$r_i^{opt} = e^{-1} \tag{5}$$

The approximation actually neglects the cost $C_{attempt}$ in regards to $\frac{C_{conn}}{T}$, and is legitimate as long as $p >> \frac{1}{m}$. Notice that when $p_i << 1$ we obtain the much simpler $T_i^{opt} \sim \frac{1}{p_i}$. We picture in Fig. 4 the evolution of $C_{Gkan}(T)$ as a function of either T or $r(T)$, for a graph having a disconnection probability $p = 1\%$.

3.2 Analysis of the heuristics

Knowing the optimality condition, we tried to estimate the performance of the Gkantsidis et al. heuristics. Let us denote by $E_{Gkan}[\Delta T_i]$ the expectation of $\Delta T_i = T_{i+1} - T_i$. The evolution of the window T under the Gkantsidis et al. heuristics leads to:

$$E_{Gkan}[\Delta T_i] = r_i - (1-r_i) \cdot \frac{T_i}{2} = r_i \left(1 + \frac{T_i}{2}\right) - \frac{T_i}{2}$$

where r_i is still given by $r_i = (1-p_i)^{T_i}$. Now, if we consider $E_{Gkan}[\Delta T_i]$ as a function of T_i, a simple study shows that this function is strictly decreasing in T_i, allowing us to introduce the following quantity:

Definition 4 (Characteristic window)**.** *At some step i of the Gkantsidis heuristics, we consider the graph G_i and its disconnection probability p_i as given. The* ***characteristic window*** *$\hat{T}_{Gkan}(p_i)$ is the positive real number that verifies:*

$$T_i < \hat{T}_{Gkan}(p_i) \iff E_{Gkan}[\Delta T_i] > 0$$

Intuitively, during the Gkantsidis heuristics, if p_i varies slowly enough, the window T_i will stay close to $\hat{T}_{Gkan}(p_i)$. Suppose for example that the disconnection probability p_i remains constant during the shuffle, so that one can define a single characteristic window $\hat{T}_{Gkan}$. Then it is clear that during the shuffle, when T_i is lower than $\hat{T}_{Gkan}$, it tends to increase : $E[T_{i+1}] > T_i$, and conversly when T_i is greater than $\hat{T}_{Gkan}$ it tends to decrease. Because of the stochastic nature of these heuristics, the variability of p_i during the shuffle, and the complexity of the dynamics ruling T_i, it appeared impossible for us to provide a formal, rigorous argument showing that the windows T_i obtained by the Gkantsidis heuristics remains close to the $\hat{T}_{Gkan}(p_i)$. However, all our experiments confirmed this fact, and the characteristic window appeared to be a very good estimator of the expectation of the window during the shuffle.

Now, to evaluate the goodness of these heuristics, one can simply, for a supposedly constant disconnection probability p, compare $\hat{T}_{Gkan}(p)$ with the optimal window $T^{opt}(p)$. The latter has already been studied above, and for the former we obtain the following bound:

$$\hat{T}_{Gkan}(p) < \sqrt{\frac{2}{p}} \tag{6}$$

Proof. Recall that

$$E_{Gkan}[\Delta T] = r(T)\left(1 + \frac{T}{2}\right) - \frac{T}{2}$$

Since $r(T) = (1-p)^T = e^{T\log(1-p)}$ and $\log(1-p) < -p$, we have:

$$E[\Delta T] < e^{-pT}(1 + \frac{T}{2}) - \frac{T}{2}$$

The convexity of the exponential function gives:

$$e^{-pT}(1 + pT) < 1$$

And finally, the fact that:

$$\frac{T/2 + 1}{1 + pT} < \frac{T}{2} \iff T > \sqrt{\frac{2}{p}}$$

shows that $T < \sqrt{\frac{2}{p}} \quad \Rightarrow \quad E[\Delta T] < 0$, which ends the proof □

When p is small, the value $\sqrt{\frac{2}{p}}$ is much smaller than the optimal $T^{opt}(p) \sim \frac{1}{p}$. This shows that the Gkantsidis et al. heuristics is too pessimistic: when the graph gets disconnected, the decrease of T is too strong; conversely, when the graph stays connected, T grows too slowly. By doing so, one obtains a very high success rate (assymptotically close to 1 when p is small), which is not the optimal (see Fig. 4).

3.3 An optimal dynamics

To improve the Gkantsidis et al. heuristics we propose the following one (with two parameters q^- and q^+):

Heuristics 2. *At the end of step i:*
IF *the graph got disconnected after the T_i edge swaps attempts*
THEN $T_{i+1} \leftarrow T_i \cdot (1 - q^-)$
ELSE $T_{i+1} \leftarrow T_i \cdot (1 + q^+)$
IF $T_{i+1} > T_{limit}$
THEN $T_{i+1} \leftarrow T_{limit}$

The constant T_{limit} represents a limit value where T doesn't need to be increased any further, even if $T_{opt} > T_{limit}$. In practice, we used $T_{limit} = 10\frac{C_{conn}}{C_{attempt}}$: at this point, using Eq. 2 and the fact that $r(T)$ is decreasing in T, we can show that the unitary cost is at most 10% over the optimal unitary cost.

Now, for the evolution of T, the main idea was to avoid the linear increase in T, which is too slow, and to allow more flexibility thanks to the two factors $1 - q^-$ and $1 + q^+$.

If we restrict ourselves to the domain where the boundary condition $T \leq T_{limit}$ doesn't apply, the expectation of ΔT_i becomes:

$$E_{new}[\Delta T_i] = q^+ r_i - q^-(1 - r_i) = r_i(q^+ + q^-) - q^- \tag{7}$$

Similarly as in Section 3.2, these heuristics have a characteristic window $\hat{T}_{new}(p)$. Since $r(T)$ and T are bijectively related, we may also use the term *characteristic success ratio* $\hat{r}_{new}$, which is the success ratio where $E_{new}[\Delta T_i] = 0$. Eq. 7 gives:

$$\hat{r}_{new} = \frac{1}{1 + \frac{q^+}{q^-}} \tag{8}$$

The discussion we developped about the Gkantsidis heuristics can be adapted here as well. Intuitively, the characteristic success ratio is an estimator of the success ratio obtained by our new heuristics. From the optimality condition found in Eq. 5, the behavior of these heuristics tends towards the optimal iff:

$$\frac{q^+}{q^-} = e - 1 \tag{9}$$

Notice that only the *ratio* $\frac{q^+}{q^-}$ is constrained by the optimality condition. The *magnitude* $\sqrt{q^+ q^-}$ can be adjusted freely. With a large magnitude, the window T becomes very unstable, and with a too small magnitude, T evolves too slowly to follow the variations of p during the shuffle. Naturally, these considerations need further investigations, which we conduced through the following empirical analysis.

3.4 Experimental evaluation of the new heuristics

To evaluate the relevance of our formal results, based on Hypothesis 1 and 2, (which is not the case, since the graph continuously changes during the shuffle) we compared empirically the three following algorithms for the adjustment of the window during the shuffle:

1. The Gkantsidis et al. heuristics (Fig. 3, Heuristics 1)
2. Our new heuristics (Heuristics 2), using the magnitude $\sqrt{q^+ q^-} = \frac{1}{10}$[6]
3. The *optimal* algorithm: at every step i, we compute the window T_i giving the minimal expectation of the unitary cost for step i. In practice, we try every possible value of T_i a certain number of times and evaluate their respective performance.[7]

The third algorithm is really the *optimal* version of the speed-up, in the sense that the window T couldn't behave better, regardless of the liability of Hypothesis 1 and 2. We compared the average unitary costs obtained with these three heuristics (respectively C_{Gkan}, C_{new} and C_{min}) for the generation of graphs with various heavy tailed[8] degree sequences. For the third algorithm, we ignored the cost of the computation of the optimal windows T_i during the shuffle. We used a wide set of parameters, and all the results were consistent with our analysis. The average window T_{Gkan} obtained with the Gkantsidis et al. heuristics behaved asymptotycally like the square root of the optimal window, leading to a much greater unitary cost, especially for topologies with low disconnection probability. On the other hand, the unitary cost obtained with our heuristics always remained at most 10% above the optimal cost.

Some typical results on heavy-tailed distributions with power-law shape of exponent $\alpha = 2.5$ and $\alpha = 3$ are shown in Table 1. These experiments show that our new heuristics is very close to the optimal. Essentially, it proves that Hypothesis 1 and 2 are legitimate in the scope of our formal analysis, or at least that their unaccuracies don't harm the behavior of our heuristics. Based on this, and from Equations 2,4,5,6, we may provide a good estimate of the unitary costs of the shuffle algorithms, for both the Gkantsidis heuristics and ours:

$$C_{gkan} = O\left(1 + \sqrt{< p >} \cdot m\right) \tag{10}$$

[6] The choice of the magnitude is difficult, but we found that $\sqrt{q^+ q^-} = 0.1$ produced the best results overall, for a wide scope of graph topologies.

[7] Note that the heavy cost of this operation prohibits its use as a heuristics, out of this context. It only serves as a reference.

[8] To obtain heavy tailed distributions, we used power-law like distributions: $P(X = k) = (k + \mu)^{-\alpha}$, where α represents the "heavy tail" behavior, while μ can be tuned to obtain the desired average z.

$\alpha = 2.5$

z	C_{Gkan}	C_{new}	C_{min}
2.1	12700	11400	11100
3	3340	2000	1930
6	481	92.0	88.2
12	32.0	3.00	2.98

$\alpha = 3$

z	C_{Gkan}	C_{new}	C_{min}
2.1	12700	11400	11100
3	3340	2000	1930
6	481	92.0	88.2
12	32.0	3.00	2.98

Table 1: Average costs of the operations performed during the shuffle for one post-validated edge swap. The benchmark is shown here for 8 graph topologies We limited ourselves to $n = 10^4$ because the computations are quite expensive, in particular concerning θ_{Gkan} and θ_{max}.

$$C_{new} = O\left(1 + < p > \cdot m\right) \tag{11}$$

(where $< p >$ is the average value of p during the shuffle). Further empirical comparisons of the two heuristics will be provided in the next section, see Table 3.

Our complexity C_{new}, despite the fact that it is asymptotically still outperformed by the complexity of the dynamic connectivity algorithm $C_{dynamic}$ (see Eq. 1), may be smaller in practice if p is small enough. For many graph topologies corresponding to real-world networks, especially graphs having a quite high density (social relations, word co-occurences, WWW), and therefore a low disconnection probability, our algorithm represents an alternative that may behave faster, and which implementation is much easier. For regular graph topologies where the low-degree vertices are rare, such as the Erdos-Reyni model [7] with high average degree z, our experiences showed that the disconnection probability p is often extremly small, sometimes even lower than $1/m$. Still, this method doesn't allow the generation of very large graphs in reasonnable time when p isn't small, which is the case of many networks topologies taken from the reality.

4 Detecting disconnections at logarithmic cost

We will now show a very simple way to detect the disconnection at low cost, thus reducing dramatically the complexity of the connectivity tests. We first outline the main idea underlying our disconnection test, then we present a modification of the naive shuffle algorithm seen in Section≈refsec:mcmc. We analyze the complexity of this new algorithm, and provide empirical results for a large span of graph topologies. Finally, to show that the unitary cost we obtain is *logarithmic*, we introduce a conjecture, strongly supported by both intuition and experiment.

4.1 Guiding principle

This paragraph has no intent to provide formal, rigorous argument, but outlines the intuitive idea that gave birth to the forthcoming study.

In realistic graph topologies, trees are unlikely: the graphs often have a significant number of *surplus* edges, *i.e.* edges that can be removed without disconnecting the graph. In other words, the ratio $\frac{m}{n}$ is often strictly greater than 1. Let us restrict ourselves to degree sequences satisfying $\frac{m}{n} > 1 + \mu$, for some positive constant μ.

We observed that, during the shuffle, disconnections were mostly caused when *small* components get separated from the main connected component. This can be intuitively explained from our hypothesis $\frac{m}{n} > 1 + \mu$. Let $C_K = v_1, \cdots, v_K$ be a set of K connected vertices. Since $\frac{m}{n} > 1 + \mu$, each vertex has an expected degree greater than $2 + 2\mu$. Now, if we suppose that the graph is random in the sense that an outgoing arc from vertex v may be branched to any vertex, independently on the other neighbors of v, and if we isolate $K - 1$ edges forming a tree core in C_K, we can expect at least $2\mu K$ *surplus* outgoing arcs from vertices in C_K. If we suppose that the giant component of our graph has a size grater than $n/2$, it seems natural to assume that each of the surplus arcs in C_K have a probability at least $1/2$ to branch to some vertex in the giant component. Therefore, the probability that C_K does not belong to the giant component is smaller than $2^{-2\mu K}$. In other words, a set of connected vertices of size K

is K-exponentially unlikely to be disconnected from the giant component. The greater K is, the more likely a graph having no components of size lower than K is connected.

4.2 A new shuffle space

We introduce the following operation:

Definition 5. *For any integer $K \leq n$, a* **K-*isolation test*** *on vertex v succeeds if and only if the connected component containing v has a size greater or equal to K.*

Implementing an isolation test of width K is straightforward; its cost is $O(K)$.

Now, recall the naive shuffle algorithm, which was a Markov chain on the set S_{CS} of the simple connected graphs having the prescribed degree sequence. The transitions of this Markov chain were algorithmically described as follows:

1. Perfom a random edge swap chosen uniformly among the $m(m-1)$ possible edge swaps
2. Test: is the graph simple?
3. Test: is the graph connected?
4. If any of the two tests fails, cancel the edge swap

Now, we propose a simple modification, where we replace the full connectivity test by two K-isolation test on the vertices concerned by the edge swap, for some integer K. For instance, if the edge swap was $(a-b),(c-d) \rightarrow (a-d),(b-c)$, one K-isolation test will be run on either a or d (since they belong to the same component) and an other on either b or c. It is easy to see that the graphs obtained with this algorithm *never* have components smaller than K. In other words this algorithm is – like the naive algorithm – a Markov chain, but the space has changed from S_{CS} to the set S_K of simple graphs with the prescribed degrees and having no components of size lower than K. Note that the graph obtained at the end has no guarantee to be connected, since $S_K \subsetneq S_{CS}$. To solve this problem, we propose a simple accept/reject approach: if the graph obtained at the end is not connected, we start over with a larger value of K, in the hope that the greater K is, the more likely the graph obtained at the the end of the shuffle will be connected. The global algorithm is sketched in Figure 5. The Markov chain on S_K is still symmetric and aperiodic, and from the reducibility of its restriction to S_{CS}, it is clear that every state in S_{CS} is reachable (since the initial state is also in S_{CS}). Thus, the distribution of the graphs obtained at the end of the algorithm is still uniform on S_{CS}.

1. Create a simple connected graph according to the prescribed degree sequence
2. Make a backup copy of the initial graph G_0
3. Set $K \leftarrow 1$
4. Set $N_{transitions}$ to the desired number of transitions to perform.
5. `REPEAT`
   . Restore the graph to its original state G_0
   . `DO` $N_{transitions}$ times:
      . Perform a random edge swap chosen uniformly among the $m(m-1)$ possible edge swaps
      . Run a simplicity test
      . Run a K-isolation test
      . `IF` at least one of the test failed `THEN` cancel the edge swap
   . Set $K \leftarrow 2K$

   `UNTIL` the graph is connected

Figure 5: Final shuffle algorithm

We can show that this algorithm ends. When $K > \frac{n}{2}$, it is easy to see that a graph having no components of size lower than K cannot have more than one component, which means that it is connected. Therefore, the algorithm must stop when K becomes greater than $\frac{n}{2}$, which happens after at most $\lfloor \log_2 n \rfloor$ iterations of the main `REPEAT` loop. Note that this is the most pessimistic case: since the $S_K, K \in \mathbb{N}$ form a decreasing sequence that converges to S_{CS}, *i.e.* $S_0 = S_1 \supset S_2 \supset \cdots \supset S_{\lfloor n/2 \rfloor + 1}$, we may hope that for a sufficiently large K, S_K becomes close enough to S_{CS} to ensure a short number of iterations. In other words, we hope that K won't reach such high values as $n/2$. The study of the expected number of iterations is made in the next section.

4.3 Complexity

We will try here to analyze the complexity of the algorithm described above. The initial operations (steps 1,2,3,4) have a linear cost $O(m)$ (see Section 2.1). At each iteration of the main `REPEAT` loop (step 5), a connectivity test and a copy of the initial graph are made: these operations also have a linear cost $O(m)$. The cost of the edge swaps, simplicity tests and K-isolation tests is $O((K+1)N_{transitions})$ per loop.

Let X be the total number of iterations of the main loop. The isolation test width follows the simple geometric sequence $1, 2, 4, \cdots, 2^X$. The total cost of the algorithm is the sum of the costs of all iterations, which is:

$$O(m) + X \cdot O(m) + \sum_{x=1}^{X} O\left((2^x + 1)N_{transitions}\right) \;=\; O\left(m(X+1) + 2^{X+1}N_{transitions}\right)$$

We saw in Section 2.1 that the number $N_{transitions}$ of transitions to perform must be at least linear: $N_{transitions} = \Omega(m)$. Moreover, as discussed in Section 2.2, the expected number of validated edge swaps will be $\Omega(N_{transitions})$, which leads to the global unitary cost:

$$C_{final} = O(2^X)$$

We will now study the expected number X of iterations of our algorithm. Recall that the set S_K is decreasing with K, with a limit S_{CS} reached for $K > n/2$. For a given degree sequence and a given isolation test width K, let us define $\epsilon(K)$ as the proportion of graphs in S_K that are not connected:

$$\epsilon(K) = \frac{|S_K - S_{CS}|}{|S_K|}$$

It is clear that $\epsilon(K)$ is decreasing in K. Now, let us consider the main algorithm at the end of the x^{th} iteration, when the $N_{transitions}$ transitions have been performed with the isolation test width $K = 2^x$, leading to a graph $G \in S_{2^x}$. If we assume that $N_{transitions}$ is large enough to ensure that G is a random element of S_{2^x}, then the probability that G is disconnected is simply $P(G \notin S_{CS})$, which is exactly $\epsilon(2^x)$.

Now, let us define an isolation test width that ensures a significant success ratio for the main iteration, *i.e.* a sufficiently low $\epsilon(K)$:

Definition 6. *The **characteristic isolation test width** $\bar{K}$ of a degree sequence is the lowest integer K such that $\epsilon(K) < \frac{1}{3}$*

The value $\frac{1}{3}$ is somewhat arbitrary: any real number lower than $\frac{1}{2}$ could have been chosen. Now, let $P_{end}(x)$ be the probability that our algorithm ends after exactly x iterations of the main `REPEAT` loop. Since $\epsilon(K)$ is decreasing in K and $1 - \epsilon(2^x)$ represents the probability that the x^{th} iteration of the main loop ends the algorithm, we obtain:

$$\forall x \mid 2^x \geq \bar{K}, \quad P_{end}(x+1) < \frac{1}{3} P_{end}(x)$$

Let us call $\bar{x}$ the lowest integer x such that $2^x \geq \bar{K}$. Considering the expected unitary cost $E[C_{final}]$ of our algorithm, since $C_{final} = O(2^X)$, we obtain:

$$E[C_{final}] \;=\; \sum_{x=1}^{\infty} 2^x \cdot P_{end}(x) \;\leq\; \sum_{x=\bar{x}}^{\infty} 2^x \left(\frac{1}{3}\right)^{x-\bar{x}}$$

Resultats coming soon

Table 2: Characteristic isolation test width for various degree sequences. Left: Erdos-Renyi. Middle: Barabasi-Albert. Right: Fab-PowerLaw

Resultats coming soon

Figure 6: Characteristic isolation test width $\bar{K}$ of the worst-case degree sequence, for fixed values of the average degree z and various sizes n. Left: $z = 2.1$. Middle: $z = 3$ Right: $z = 10$

which gives:

$$E[C_{final}] \ \leq \ 3 \cdot 2^{\bar{x}} \ \leq \ 6\bar{K} \ = \ O(\bar{K})$$

4.4 Characteristic isolation test width

The only bounds we obtained so far for $\bar{K}$ are $1 \leq \bar{K} \leq 1 + \lfloor n/2 \rfloor$. Following the ideas described in Section 4.1, and based on empirical evidence, we have strong reasons to believe that $\bar{K}$ is actually much lower than $\frac{n}{2}$, at least for non-tree topologies like the ones described in Section 4.1 that verify $\frac{m}{n} > 1+\mu$. Nevertheless, providing a good upper bound is not easy: we observed that $\bar{K}$ depends strongly on the degree sequence, and that among graphs having similar degree distributions, the larger graphs have larger $\bar{K}$. Empirical results are presented in Tables 2, for a set of different graph topologies.

Moreover, we tried to imagine what could be the worst degree sequence, *i.e.* the degree sequence having the largest $\bar{K}$, for a given average degree z. It appeared that for a fixed average degree z, the realizable[9] sequence having as many '2' as possible is a good candidate: it is the worst case we found. We plotted in Figure 6 the characteristic isolation test width of our "worst-case" sequences as a function of their size n, and for various values of their mean z.

The coherence of the empirical results we obtained, combined with the intuition described in Section 4.1, led us to the following conjecture:

Conjecture 1. *There exists a real function η such that, for any degree sequence of size n and mean z, the characteristic isolation width $\bar{K}$ verifies:*

$$\bar{K} \leq \eta(z) \log n$$

According to this conjecture, and if we restrict ourselves – as discussed before – to graph topologies verifying the hypothesis $\frac{m}{n} > 1+\mu$, our algorithm has a logarithmic unitary cost $O(\log n)$. It outperforms the shuffle based on the best dynamic connectivity algorithms known so far (see Eq. 1).

5 Conclusion

Focusing on the speed-up method introduced by Gkantsidis et al. for the Markov chain Monte Carlo algorithm, we introduced a formal background allowing us to show that this heuristic is not optimal in its own family. We improved it in order to reach the optimal, and empirically confirmed the results. Our heuristic may also be used for other problems based on Markov chain Monte Carlo algorithms, without losing its optimal properties.

Going further, and focusing on the generation of random conected graphs, we then introduced an original method allowing to shuffle non-tree connected graphs (*i.e.* connected graphs having an average

[9] In this paper, a degree sequence is said *realizable* when there exists at least one simple connected graph with this degree sequence

coming soon

Table 3: Cout theoriques et Temps d'execution des differents algos

degree strictly greater than 2) in logarithmic time per edge swaps, or per transition. It outperforms the previous best known methods, and has the advantage of being extremly easy to implement. Moreover, the asymptotical complexity constants remain extremely low on most graph topologies. We provide an implementation of this last algorithm [25]. Note however that the last result relies on a conjecture that we were unable to prove, but for which we provided strong empirical evidence.

References

[1] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. *Proc. of the 32nd ACM STOC*, pages 171–180, 2000.

[2] R. Albert and A. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47, 2002.

[3] B. Bollobas. *Random Graphs*. Academic Press, London - New York, 1985.

[4] A. Clauset and C. Moore. Traceroute sampling makes random graphs appear to have power law degree distributions. *cond-mat/0312674*, 2003.

[5] S.N. Dorogovtsev and J.F.F. Mendes. Evolution of networks. *Adv. Phys.*, 51:1079, 2002.

[6] P. Erdos and T. Gallai. Graphs with prescribed degree of vertices. *Mat. Lapok*, 11:264–274, 1960.

[7] P. Erdös and A. Rényi. On random graphs. *Publ. Math. Debrecen*, 6:290–291, 1959.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *Proc. ACM SIGCOMM*, 29:251–262, 1999.

[9] C. Gkantsidis, M. Mihail, and E. Zegura. The markov chain simulation method for generating connected power law random graphs. *Proc. of ALENEX'03, LNCS*, pages 16–25, 2003.

[10] S. L. Hakimi. On the realizability of a set of integers as degrees of the vertices of a linear graph. *SIAM Journal*, 10(3):496–506, 1962.

[11] V. Havel. A remark on the existence of finite graphs. *Caposis Pest. Mat.*, 80:496–506, 1955.

[12] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[13] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Proc. of the 30th ACM STOC*, pages 79–89, 1998.

[14] M. Jerrum and A. Sinclair. Approximating the permanent. volume 18, pages 1149–1178, 1989.

[15] J. M. Roberts Jr. Simple methods for simulating sociomatrices with given marginal totals. *Social Networks*, 22:273–283, 2000.

[16] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. Uniform generation of random graphs with arbitrary degree sequences. *submitted to Phys. Rev. E*, 2001.

[17] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, pages 161–179, 1995.

[18] M. Molloy and B. Reed. The size of the giant component of a random graph with a given degree sequence. *Combinatorics, Probability and Computing*, 7:295, 1998.

[19] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[20] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64(026118), 2001.

[21] A. R. Rao, R. Jana, and S. Bandyopadhyay. A markov chain monte carlo method for generating random (0,1)-matrices with given marginals. *Indian Journal of Statistics*, 58(A):225–242, 1996.

[22] Alexandre O. Stauffer and Valmir C. Barbosa. A study of the edge-switching markov-chain method for the generation of random graphs. In *preprint: arXiv:cs.DM/0512105*, 2006.

[23] R. Taylor. Constrained switchings in graphs. *Combinatorial Mathematics*, 8:314–336, 1980.

[24] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. *Proc. of the 32nd ACM STOC*, pages 343–350, 2000.

[25] www.liafa.jussieu.fr/∼fabien/generation.

A Evaluation of the bias of the common method

The "common method" to generate random simple connected graphs with a prescribed degree sequence is the following :

1. Generate a graph G with the Molloy and Reed model [17].
2. Remove the multiple edges and loops, obtaining a simple graph G_S.
3. Keep only the largest connected component, obtaining a simple connected subgraph G_{CS}

In the following, we also call G_C the subgraph obtained by step 3 without step 2 (G_C is the non-simple giant connected component of G). It is clear that G_S, G_C and G_{CS} are different from G. We provide here experimental evidences that this difference is significant. Since our model doesn't suffer of any such bias, as it is simple and connected from the beginning, we recommend its use for anyone who needs to generate random simple connected graphs with a prescribed degree sequence.

Notations

We call N the number of vertices in G, M the number of edges and Z the average degree. Likewise, N_C, N_S, N_{CS}, M_C, M_S, M_{CS}, Z_C, Z_S and Z_{CS} refer respectively to G_C, G_S and G_{CS}.

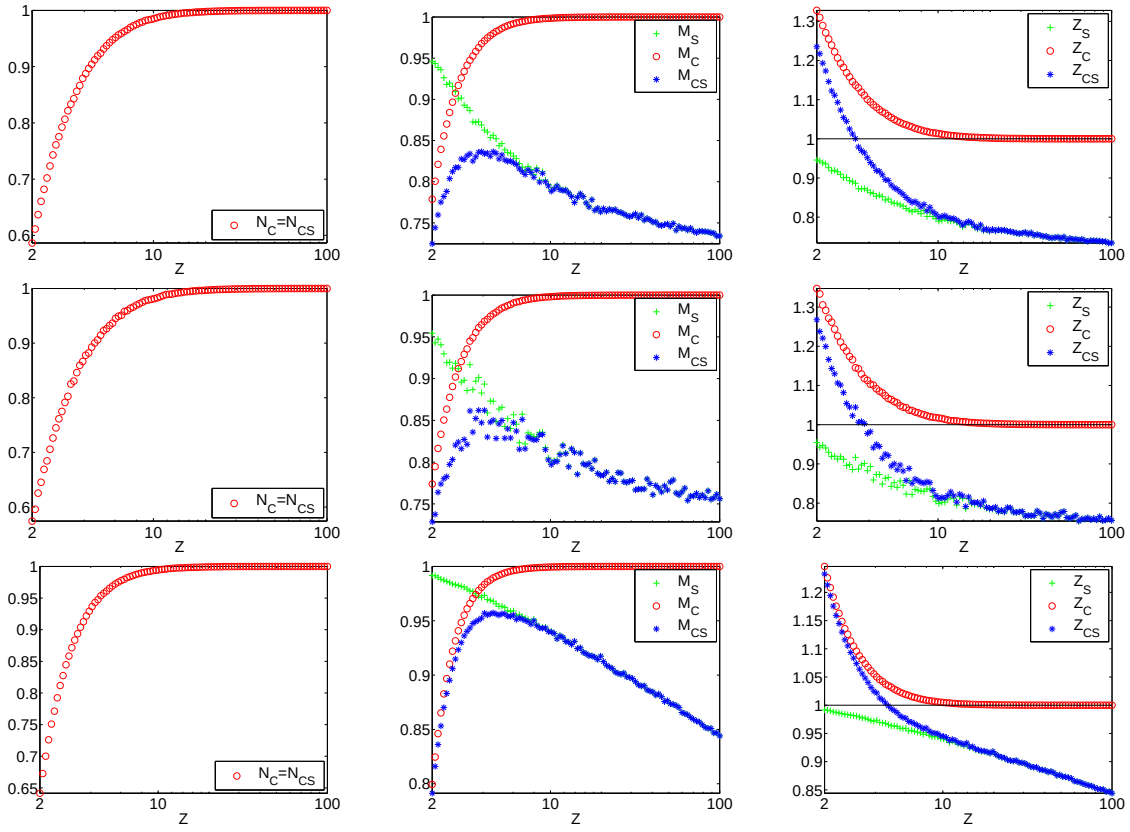

Figure 7: Comparison of the number of vertices (left), the number of edges (center) and the average degree (right) in the graphs G_S, G_C and G_{CS}, for various values of the average degree Z of the original graph G. We used heavy-tailed degree distributions with $N = 10^4, \alpha = 2.1$ (top), $N = 10^5, \alpha = 2.1$ (middle) and $N = 10^4, \alpha = 2.5$ (bottom).

Plots

To quantify the modifications caused by the removal of multiple edges and/or the restriction to the giant connected component, we plotted the number of vertices, the number of edges and the average degrees of the concerned subgraphs G_C, G_S and G_{CS} against the average degree of G. In each plot, the three curves refer to G_C (red circles), G_S (green plus) and G_{CS}(blue stars). The quantities are **normalized** so that a value of 1 represents the value of the concerned quantity in G.

Notice that, since N_S is always equal to N (the removal of edges by itself does not change the number of vertices), we only plotted N_C, which is also equal to N_{CS}.

Discussion

Many things can be observed from those plots. In particular :

- The left and middle plots show clearly that one loses a significant part of the graph when performing multiple edge removal, restricting to the giant component, or both.
- The similarity between the plots at the top and in the middle show that the size N has very little, if any, influence on this loss. The only noticeable difference comes fom the fact that the top plots, due to their lower computation costs, were averaged on more instances than the middle ones.
- The bottom plots are closer to 1, meaning that the bias is less significant. This is due to the greater exponent α, causing the heavy-tailed degree distribution to be less heterogenous. Thus, less vertices have very low degree (these ones get more likely removed in G_C) or very high degree (these ones are more likely to get many edges removed in G_S).
- The left part of the plots (low average degree Z) show a significant loss of vertices in G_C. This is of course because the more edges we have, the bigger the giant connected component is. On the other hand, the right part of the plots (high average degree Z) show an increasing loss of edges due to the removal of more multiple edges.
- The plots on the right-hand side show that two opposite biases act on the average degree Z_{CS} of G_{CS}: the multiple edges removals tends to lower it, while the removal of vertices that don't belong to the giant component tends to raise it (since these vertices more likely have a low degree).

Conclusions

We showed that the bias caused by the two last steps of the "common method" is significant, not only on the size of the graph but also on its properties, like the average degree. These biases should therefore cause the deviation of many other properties. Our model, which respect exactly the degree sequence given at the beginning, represents a reference that may be used to better quantify these deviations. Its simplicity and efficiency should also convince users to implement it (or to use our implementation, available at [25]). Notably, it provides an easy way to separate the properties of the known models, like the Barabàsi-Albert one, in two groups: the ones that come from the degree distribution only, and the ones that come from the model itself.

B Disconnection probability

For a given connected graph G, the disconnection probability p_G is the probability, that a random edge swap searates G in two components. One key hypothesis for the formal study of the optimal window problem is that p remains constant during the shuffle. We will show here that this hypothesis is only an approximation, but that the error induced by the variations of p have a small impact on the performance of our heuristics.

To estimate p_G, we used a brute force method: we perform a random edge swap on G, test wether G is still connected, and start over until our estimation of p_G converges. As we write these lines, a very recent paper [22] just proposed an original way to directly measure p in linear time, thus reducing dramatically the cost of measuring p; however, the simulations we made so far seemed accurate enough for our purpose.

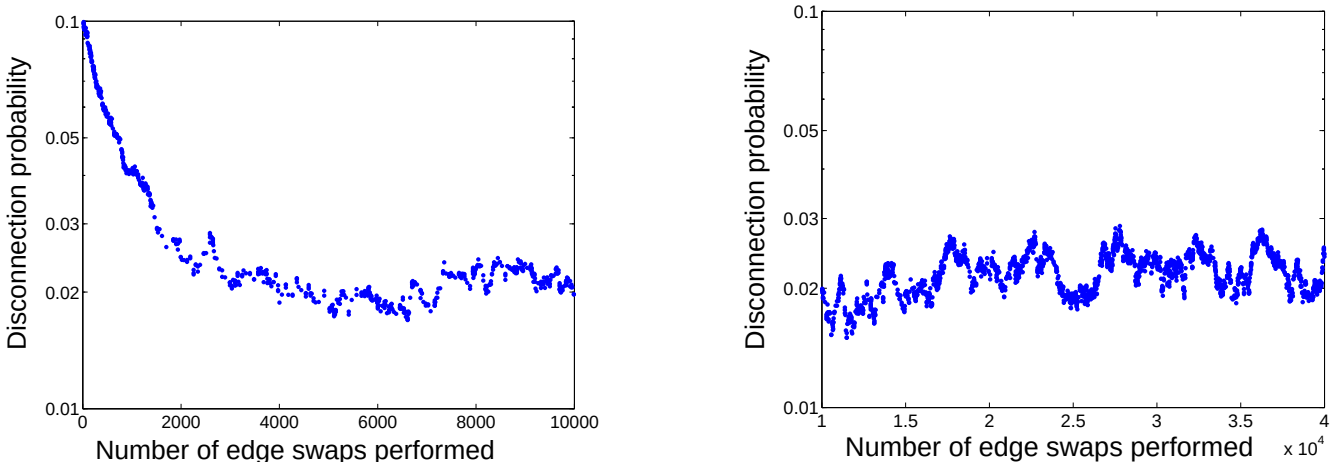

Figure 8: Evolution of the disconnection probability during the shuffle

We measured p during the whole shuffle process. Some typical results are shown in Figure 8, the graph used here had $n = 1000$ vertices, an average degree $z = 6$, and a heavy-tail degree distribution of exponent $\alpha = 2.5$. We observe two different phenomenons:

1. At the beginning of the shuffle, we observe a slow decrease of the p (in average) until it reaches its average value, here $p_{avg} \approx 0.02$. This is because of the second step of the generation process (see Section 2), where we connect the graph by joining each of its component to the giant one. These components, being attached to the rest of the graph with only one edge, are likely to get disconnected. Thus, the initial state of the graph is strongly biased, and the shuffle slowly decreases p as G becomes more and more random.
2. After this initial decrease (which lasts during roughly m edge swaps), the disconnection probability doesn't converge, but oscillate around its average value p_{avg}. The amplitude of these oscillations are not neglectable, but they stay within a factor 2 from the average value.

Now, let us estimate the impact of the variability of the disconnection probability on the efficiency of our heuristics. Let us place at some step of the shuffle process. Say the graph G has a disconnection probability p, and say that our estimate of p is p_{est}. Let $C(p_{est})$ be the theretical unitary cost of the suffle when the window is set to the estimated optimal $T^{opt}(p_{est}) = -\log(1-p_{est})^{-1}$, as given in Eq. 4. From our discussion about the optimal window in Section 3.1, it is clear that $C(p)$ is minimal when the estimate is perfect: $p_{est} = p$. We plotted in Figure 9 the ratio between the cost obtained with a perfect estimate and the cost obtained with a bad estimate, as a function of the deviation p_{est}/p of the estimate. We also plotted two vertical lines that correspond to two worst-case deviations, inspired by the empirical evolution of p we showed above: say that p_{min} and p_{max} are respectively the all-time minimum and maximum of p during the shuffle, the vertical lines correspond to $\frac{p_{est}}{p} = \frac{p_{min}}{p_{max}}$ and to $\frac{p_{est}}{p} = \frac{p_{max}}{p_{min}}$. Even in those very pessimistic cases, the efficiency loss is less than 20%. This explains the good results we

obtained with our heuristics, althought many formal analysis were based on approximations. Let us add that the case shown here is representative of the general case: we observed a similar – if not better – behavior when working on other graph topologies.

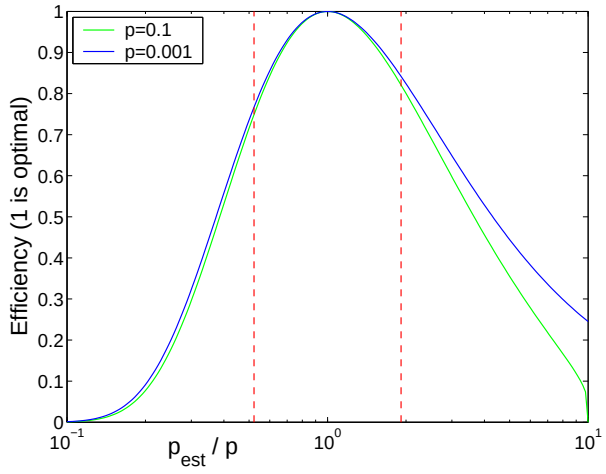

Figure 9: Relation between efficiency loss and error made when estimating p