Random Generation of Deterministic Acyclic Automata using the Recursive Method*

Sven De Felice[1] and Cyril Nicaud[1]

LIGM, Université Paris-Est, 77454 Marne-la-Vallée Cedex 2, France
defelic@univ-mlv.fr, nicaud@univ-mlv.fr

Abstract. In this article, we propose a uniform random generator for accessible deterministic acyclic automata with n states, which is based on the recursive method. The generator has a preprocessing that requires $\mathcal{O}(n^3)$ arithmetic operations, and, once it is done, can generate acyclic automata using $\mathcal{O}(n)$ arithmetic operations for each sample. We also propose a lazy version of the algorithm that takes advantage of the typical shape of random acyclic automata to reduce experimentally the preprocessing. Using this algorithm, we provide some statistics on acyclic automata with up to 1000 states.

1 Introduction

The field of random generation has been very active in the last two decades, developing general techniques based on combinatorics and probabilities to produce efficient random samplers for combinatorial structures that appear in computer science. The main focus is to build an algorithm for a given combinatorial set E, i.e. a set together with a size function, which takes an integer n as input and produces an element of E of size n uniformly at random. The interest in random generators comes from both practice and theory: they are commonly used as an alternative to benchmarks, in order to test the efficiency of implementations; they are also widely used by theorists in their works on describing the typical properties of large random objects, which is a cornerstone in average case analysis of algorithms [7].

This article deals with the random generation of acyclic deterministic automata with n states on a fixed finite alphabet. Acyclic automata are automata recognizing finite languages, and, as such, form an important subclass of automata. They are especially used in applications such as in linguistics where the languages of interest are essentially finite. A better understanding of their combinatorics, of their typical behavior and the average cases analysis of algorithms that handle that kind of structures is a long term goal of our work.

In random generation standards, we aim at designing algorithms that can produce experimental statistics on objects of size at least 1000, in a reasonable

* This work is supported by the French National Agency (ANR) through ANR-10-LABX-58, through ANR-2010-BLAN-0204 and through ANR-JCJC-12-JS02-012-01

amount of time: though probabilistic and combinatorial in nature, random generator are above all algorithms and the main concern when designing them is to optimize their complexity, in order to allow the generation of objects that are large enough.

In the sequel, we present how deterministic acyclic automata can be generated using a technique known as the *recursive method*. It is based on an inductive specification of the objects of interest and consists in two steps: a preprocessing which is costly but need to be done only once, and the random generation itself. This method originates in [10] and was systematized in [8] to classes of combinatorial structures that can be described using the symbolic method [7, Ch. I]. Acyclic automata cannot be described in such a way, and the aim of this article is to show that the technique applies anyway, though not automatically. Our algorithm has a preprocessing step using $\mathcal{O}(n^3)$ arithmetic operations, and can then generate deterministic acyclic automata with n states using a linear number of arithmetic operations. We also propose an improved version that is very efficient in practice to lower the complexity of the preprocessing. This efficiency relies on the typical shape of a random acyclic automaton: we formulate its complexity as a function of a parameter of its output (its width, defined in Section 4.5) which seems to grow very slowly in practice. Though requiring a detailed probabilistic analysis for this observation to be mathematically established, we were able to compute some statistics on automata with up to 1000 states within a few minutes using our improved algorithm. This is an example where information on typical structures helps in designing better algorithms.

One of our motivation for studying acyclic automata comes from the theory of *automaton groups*. Consider a letter-by-letter deterministic transducer, and the different functions from A^* to A^* it realizes when taking all the possibilities for the initial state. Assume that some conditions ensure that this functions are permutations of A^* and consider the group they generate. That kind of groups are especially rich and have been studied by both mathematicians and computer scientists (see [1] for more details). In particular, Antonenko [3] gave some conditions on the shape of the transducer that ensure that the generated group is finite, and these conditions make use of acyclic automata. The quantitative study of Antonenko automata therefore requires more knowledge on deterministic acyclic automata, and could be a first step toward understanding the combinatorics behind that kind of groups.

Related works. In [4], the first author and Carnino proposed a solution to the same problem based on Markov chain techniques. This yields an elegant and easily adaptable algorithm[1], that produces an acyclic automaton with almost uniform distribution. The major drawback of this method is that there is no estimation of the bias induced when halting the algorithm after a given number of iterations, making the design of the algorithm difficult when uniformity is important[2].

[1] For instance, they described a Markov chain on minimal acyclic automata only.

[2] Using Markov chain terminology: the mixing time of the chain is not known and seems to be difficult to estimate.

The recursive method is based on the combinatorial properties of the objects to be generated. In [9], Liskovets presented inclusion-exclusion results on the number of deterministic acyclic automata and on some related quantities. We will use his work at several stages of the design of our algorithm. Prior to this work, Domaratzki, Kisman and Shallit proposed lower and upper bounds on the number of acyclic automata in [5, 6].

In [2], Almeida, Moreira and Reis gave a solution to a related problem: they proposed an algorithm that generate *exhaustively* all minimal automata with a given number of states. Because the number of minimal acyclic automata grows very fast, such an exhaustive generator is limited to small number of states (there are more than $7 \cdot 10^{14}$ minimal acyclic automata with 15 states on a two-letter alphabet), but it has the virtue of checking every object unlike a random generator. Both solutions are relevant, in different manners, when testing conjectures.

2 Definition and Notations

For any integer $n \geq 1$, let $[n] = \{1, \ldots, n\}$ be the set of integer from 1 to n. If n and m are two non-negative integers, we denote by $\mathrm{Surj}(n, m)$ the number of surjections from $[n]$ onto $[m]$. The value of $\mathrm{Surj}(n, m)$ can be computed recursively using the formula: $\mathrm{Surj}(n, m) = m \cdot \mathrm{Surj}(n-1, m-1) + m \cdot \mathrm{Surj}(n-1, m)$, with initial conditions $\mathrm{Surj}(n, 1) = 1$ and $\mathrm{Surj}(n, m) = 0$ if $m > n$.

Let A be a finite alphabet, a *deterministic automaton* (or *automaton* for short) on A is a tuple (Q, δ, q_0), where Q is a finite set of *states*, δ is the *transition function*, a possibly partial mapping from $Q \times A$ to Q, and $q_0 \in Q$ is the *initial state*. If $p, q \in Q$ and $a \in A$ are such that $\delta(p, a) = q$, then (p, a, q) is the *transition* from p to q labelled by a, and is denoted by $p \xrightarrow{a} q$. An automaton $\mathcal{A} = (A, Q, \delta)$ is classically seen as a labelled directed graph whose set of vertices is Q and whose edges are the transitions of $\mathcal{A}$.

An automaton is *accessible* (or *initially connected*) when for every state p there exists a path starting from the initial state that ends at p. The transition function is extended to $Q \times A^*$ by morphism, setting $\delta(p, \varepsilon) = p$ for every $p \in Q$ and $\delta(p, ua) = \delta(\delta(p, u), a)$ when everything is defined, and undefined otherwise.

An automaton is *acyclic* when its graph is acyclic. A *source* of an automaton $\mathcal{A}$ is a state with no incoming transition. An acyclic automaton always has at least one source, and accessible acyclic automata are acyclic automata with exactly one source.

In the sequel, the set of states of an automaton with n states will almost always be $[n]$. If $\mathcal{A}$ is an automaton of set of states $[n]$ and X is a set of n positive integers, the *relabelling of $\mathcal{A}$ using X* is the automaton obtained from $\mathcal{A}$ when changing the states labels by elements of X, while respecting their relative order: if $p < q$ in $\mathcal{A}$ then the new label of p is smaller than the one of q. Notice that there is only one way to do so.

Important : We are not interested in final states except in the experimental section of this article (Section 5). We will therefore call "automaton" a determin-

istic automaton without final states throughout the article, and denote classical deterministic automata by "automata with final states".

3 Combinatorics of Acyclic Automata

3.1 Liskovet's formula

In [9], Liskovets establishes formulas to count the number of acyclic automata. These results relies on the inclusion-exclusion method [11], which is a classical and elegant technique yielding formulas with alternating sums. One such result is the following (set $r = 1$ in Eq. (3) of [9]): if $a_k(n)$ denote the number of labelled acyclic automata with n states on a k-letter alphabet, then

$$a_k(n) = \sum_{t=0}^{n-1} \binom{n}{t} (-1)^{n-t-1} (t+1)^{k(n-t)} a_k(t). \tag{1}$$

Let $\alpha_k(n, s)$ denote the number of labelled acyclic automata with n states on a k-letter alphabet that have exactly s sources. Using almost the same proof as Liskovets' one can obtain the following formula:

$$\alpha_k(n, s) = \binom{n}{s} \sum_{i=0}^{n-s} \binom{n-s}{i} (-1)^i a_k(n-s-i) \cdot (n-s-i+1)^{k(s+i)}. \tag{2}$$

Lemma 1. *The values of $a_k(m)$ for every $m \in [n]$ can be computed using $\mathcal{O}(n^2)$ arithmetic operations and storing $\mathcal{O}(n^2)$ integers. The values of $\alpha_k(m, t)$ for every $m \in [n]$ and $t \in [s]$, with $s \leq n$, can be computed using $\mathcal{O}(sn^2)$ arithmetic operations and storing $\mathcal{O}(n^2)$ integers.*

Though originating from a combinatorial description, inclusion-exclusion formulas are not always very useful when designing efficient random generators[3], because of the complications inherent to the exclusions of subsets, which correspond to the minus signs in the formulas.

We will however use Liskovets' formulas later in our algorithms, as a shortcut to compute the required values more efficiently. For now, we need a more straightforward combinatorial decomposition that is amenable to the recursive method; this is the purpose of next section.

3.2 Decomposition using sources and secondary sources

Our decomposition consists intuitively in repeatedly pruning the automaton by removing its sources. This is a classical idea coming from the enumeration of acyclic directed graphs. If $\mathcal{A}$ is an acyclic automaton, a state of $\mathcal{A}$ is a *secondary source* if it is not a source and if all its incoming transitions come from sources.

[3] There is a noticeable exception when rejection techniques can be applied, but it does not appear to be the case for acyclic automata.

In other words, p is a secondary source if it has no more incoming transition when the sources of $\mathcal{A}$ are removed.

Let $\mathcal{A}$ be an acyclic automaton with n states and s sources on a k-letter alphabet; when the s sources and their outgoing transitions are removed from $\mathcal{A}$, what remains is an acyclic automaton $\mathcal{B}$ with $n-s$ states. We obtain a formula by partitioning the possibilities according to the number u of sources of $\mathcal{B}$, that is, the number of secondary sources of $\mathcal{A}$. Thinking backward, for any given $\mathcal{B}$ with $n-s$ states and u sources, one can reconstruct an automaton $\mathcal{A}$ by doing the following:

1. Choose the set of labels $Y \subseteq [n]$ for the s sources of $\mathcal{A}$ and relabel $\mathcal{B}$ following $[n] \setminus Y$.
2. For every transition starting from one of the s sources, choose whether it is undefined or not, and if it is not, where it ends amongst the $n-s$ possibilities. This must be done in such a way that every secondary source has at least one incoming transition.

There are $\binom{n}{s}$ ways to choose the set of labels. Let $\beta_k(n,s,u)$ be the number of possibilities for the second item. We count the number of valid configurations for the ks transitions starting from a source using the number i of transitions that end in a secondary source as a parameter: at fixed i, a valid configuration is obtained by choosing the i transitions amongst the ks possibilities, how they are mapped to their u possible ending states in a surjective way (since each secondary source must have an incoming transition from a source) and the ending state of each of the $ks-i$ remaining transitions, which can be either undefined or one of the $n-s-u$ states that are neither a source nor a secondary source. Hence, the number of valid configurations is given by:

$$\beta_k(n,s,u) = \sum_{i=u}^{ks} \binom{ks}{i} \cdot \mathrm{Surj}(i,u) \cdot (n-s-u+1)^{ks-i}. \tag{3}$$

This yields the following formula for the number of acyclic automata with $s < n$ sources:

$$\alpha_k(n,s) = \sum_{u=1}^{\min(ks,n-s)} \binom{n}{s} \cdot \beta_k(n,s,u) \cdot \alpha_k(n-s,u), \tag{4}$$

since there are $\alpha_k(n-s,u)$ ways to choose $\mathcal{B}$. Of course, we also have $\alpha_k(n,n) = 1$.

Remark that for computational purposes, Eq. (4) is not as good as Liskovet's formula, which can be computed in time $\mathcal{O}(n^2)$ according to Lemma 1. The gain is the combinatorial description that can be directly turned into a random generator, provided all the required quantities are already computed.

3.3 Another description for $\beta_k(n,s,u)$

For $\mathcal{T}$, $\mathcal{U}$ and $\mathcal{R}$ three finite sets such that $\mathcal{U}$ and $\mathcal{R}$ are disjoint, consider the family $\mathcal{G}(\mathcal{T},\mathcal{U},\mathcal{R})$ of partial functions from $\mathcal{T}$ to $\mathcal{U} \cup \mathcal{R}$ such that every $q \in \mathcal{U}$

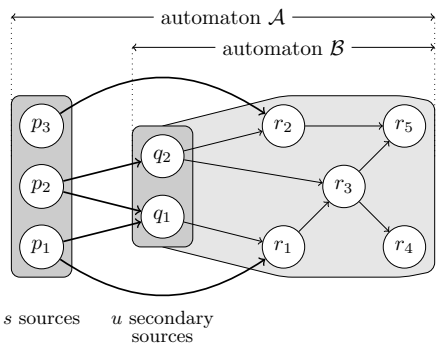


Fig. 1. Main decomposition on an automaton with $n = 10$ states on a two-letter alphabet: To build an acyclic automaton $\mathcal{A}$ with $s = 3$ sources and $u = 2$ secondary sources, one has to choose $\mathcal{B}$, an acyclic automaton with 2 sources, and to set the transitions starting from the sources of $\mathcal{A}$: they can either be defined or undefined, cannot end in a source of $\mathcal{A}$, and must cover all the sources of $\mathcal{B}$. The number of ways to do so is $\beta_2(10, 3, 2) = \gamma(6, 2, 5)$.

has at least one preimage. Let $\gamma(t, u, r)$ be the cardinality of $\mathcal{G}(\mathcal{T}, \mathcal{U}, \mathcal{R})$ when $|\mathcal{T}| = t$, $|\mathcal{U}| = u$ and $|\mathcal{R}| = r$.

Recall that $\beta_k(n, s, u)$ counts the number of ways to define the transitions starting from the s sources such that each of the u secondary sources has at least one incoming transition, with a total of n states. Let $\mathcal{T}$ denote the pairs (s, a) where s is a source and a is a letter, let $\mathcal{U}$ denote the set of secondary sources, and let $\mathcal{R}$ denote the set of states that are neither a source nor a secondary source. The function that maps each (s, a) to its ending state, when it exists, is a partial function from $\mathcal{T}$ to $\mathcal{U} \cup \mathcal{R}$ such that every $q \in \mathcal{U}$ has at least one preimage: it is therefore an element of $\mathcal{G}(\mathcal{T}, \mathcal{U}, \mathcal{R})$. Conversely, every such function corresponds to a valid choice for the transitions starting from a source. Hence we have the identity $\beta_k(n, s, u) = \gamma(ks, u, n - s - u)$.

The inductive description of $\mathcal{G}(\mathcal{T}, \mathcal{U}, \mathcal{R})$ is the following. Let x be any element of $\mathcal{T}$. The functions in $\mathcal{G}(\mathcal{T}, \mathcal{U}, \mathcal{R})$ fall in two categories: those such that x is the unique preimage of an element of $\mathcal{U}$ and the other ones. A function of the first category restricted to $\mathcal{T} \setminus \{x\}$ is exactly an element of $\mathcal{G}(\mathcal{T} \setminus \{x\}, \mathcal{U} \setminus \{q\}, \mathcal{R})$: there are $u \cdot \gamma(t - 1, u - 1, r)$ such possibilities. Otherwise, the restriction to $\mathcal{T} \setminus \{x\}$ is exactly an element of $\mathcal{G}(\mathcal{T} \setminus \{x\}, \mathcal{U}, \mathcal{R})$ and there are $(u + r + 1)\gamma(t - 1, u, r)$ possibilities, $u + r - 1$ being the number of different ways to choose the image of x including the case when it is undefined. We therefore obtained that:

$$\gamma(t, u, r) = u \cdot \gamma(t - 1, u - 1, r) + (r + u + 1) \cdot \gamma(t - 1, u, r). \tag{5}$$

Moreover this formula has a combinatorial meaning, since it is a discussion on the different possibilities for the image of a given element of $\mathcal{T}$. The boundary conditions are: $\gamma(t, 0, r) = (r + 1)^t$ and $\gamma(t, u, r) = 0$ for $t < u$.

3.4 Remark on labelled combinatorial structures

The combinatorial study of labelled structures, i.e. when the n vertices are labelled with the elements of $[n]$, is often easier than the one of unlabelled structures, since symmetries that can appear in the unlabelled case usually make the counting more complicated.

The situation is different for structures that are *rigid*, that is, structures with no symmetry[4], since the number of labelled structures is exactly $n!$ times the number of unlabelled ones. Fortunately, this is the case for deterministic automata when they are accessible, as explained in [9]. In particular, the number of unlabelled accessible acyclic automata is $\frac{1}{n!}\alpha_k(n,1)$ and they can be randomly generated as labelled structures and still being uniform as unlabelled automata.

4 Random Generator

4.1 The recursive method

As stated in the introduction, acyclic automata cannot not be directly described using the symbolic method [7, Ch. I]. Therefore, we cannot use the automatic translation into a random generator proposed in [8]. The purpose of this section is to adapt the method to our specific formulas of Section 3 in order to get such a generator. Remark informally that our formulas are always of the form $\lambda_n = \sum_i \lambda_{n,i}$, where parameter i has a combinatorial meaning. Assume that λ_n and all the $\lambda_{n,i}$ have already been computed, it is then easy to generate the value of parameter i for a uniform object of size n, since $\mathbb{P}_n(\text{parameter} = i) = \frac{\lambda_{n,i}}{\lambda_n}$. The idea is to choose i with correct probability, reducing the problem to the uniform random generation of smaller objects. Some additional constructions can be required to finally build the result, depending on the combinatorial construction that leads to the formula for λ_n.

4.2 Application to acyclic automata

The method described above can directly be applied to generate uniformly at random accessible acyclic automata. The first unoptimized version of the algorithm for an acyclic automaton with s sources is the following:

1. Compute all the values of $a_k(m)$, $\alpha_k(m,s)$, $\beta_k(m,s,u)$ and also of $\binom{ks}{i}$ and $\mathrm{Surj}(i,u)$ for every $m \in [n]$, every $s \in [m]$, every $u \in [m-s]$ and every $i \in [m]$.
2. Use Eq. (4) with s sources to generate the value of u with correct probability: The number of secondary sources takes value u with probability
$$\frac{\binom{n}{s} \cdot \beta_k(n,s,u) \cdot \alpha_k(n-s,u)}{\alpha_k(n,s)}. \tag{6}$$
3. Recursively generate an acyclic automaton $\mathcal{B}$ of size $n-s$ having u sources.
4. Choose the set of source labels X, and relabel $\mathcal{B}$ following $[n] \setminus X$.
5. Use Eq. (3) to generate the number of transitions starting from a source and ending in a secondary source. The number of such transitions takes value i with probability
$$\frac{\binom{ks}{i} \cdot \mathrm{Surj}(i,u) \cdot (n-s-u+1)^{ks-i}}{\beta_k(n,s,u)}. \tag{7}$$

[4] Formally, the group of structure automorphisms is trivial.

6. Generate the transitions starting from sources with correct probability, by choosing the i transitions ending in a secondary source, the surjective way they are associated to secondary sources, and by choosing the ending state of the other ones (or whether they are undefined).

Lemma 2. *The method above produces a random acyclic automaton with n states and s sources uniformly at random.*

The proof is done by induction on n and follows from the unicity of our decomposition. A direct computation of the probabilities using Eq. (6), Eq. (7), the induction hypothesis and the probability associated with step 6 yields that $\mathcal{A}$ is produced with uniform probability.

This straightforward way to turn the formulas of Section 3 into a random generator is constitutive of the recursive method. Notice that a random generator for accessible acyclic automata of size n is obtained by setting $s = 1$.

Also remark that the first step is the main limitation of this method, since it requires quite some time and space to compute and store all the needed results. Using Eq. (3) and Eq. (4) to perform the computations require $\mathcal{O}(n^4)$ arithmetic operations. However, it is important to notice that this preprocessing must be done only once. Thereafter, as will be explained in Section 4.4, the random generation of an acyclic automaton with n states is done in time $\mathcal{O}(n)$.

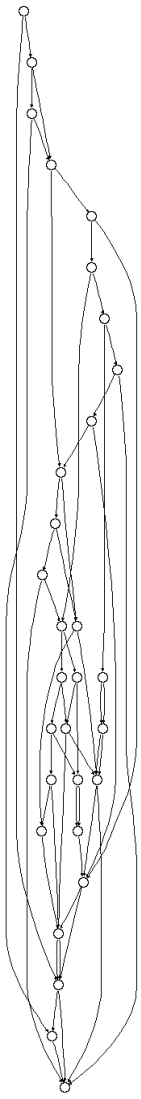
In the sequel we will show how to improve the complexity of the algorithm.

4.3 Using γ instead of β

In Section 3.3 is explained that $\beta_k(n, s, u) = \gamma(ks, u, r)$, where both quantities describe how to link the sources to the secondary sources, in two different ways. The formula for $\gamma(ks, u, r)$ is more advantageous in terms of time complexity, since one can compute all the $\mathcal{O}(n^3)$ needed values for γ using $\mathcal{O}(n^3)$ arithmetic operations with Eq. (5). Using γ instead of β_k, we also do not need to compute the values of $\mathrm{Surj}(i, u)$ anymore, since the combinatorial decomposition that leads to Eq. (5) can be directly turned into an algorithm: in order to generate a random element f of $\mathcal{G}(\mathcal{T}, \mathcal{U}, \mathcal{R})$, start from any $x \in \mathcal{T}$ then pick a random number d in $[\gamma(t, u, r)]$. If $d \leq u \cdot \gamma(t-1, u-1, r)$, choose uniformly an element q of $\mathcal{U}$ and set that x is the unique preimage of q by f; it remains to recursively draw the restriction of f to $\mathcal{T} \setminus \{x\}$ in $\mathcal{G}(\mathcal{T} \setminus \{x\}, \mathcal{U} \setminus \{q\}, \mathcal{R})$. If $d > u \cdot \gamma(t-1, u-1, r)$, choose uniformly the image of x by f in $\mathcal{U} \cup \mathcal{R} \cup \{\perp\}$, where $f(x) = \perp$ means that $f(x)$ is undefined, and recursively draw the restriction of f to $\mathcal{T} \setminus \{x\}$ in $\mathcal{G}(\mathcal{T} \setminus \{x\}, \mathcal{U}, \mathcal{R})$. The complexity of generating an element of $\mathcal{G}(\mathcal{T}, \mathcal{U}, \mathcal{R})$ is therefore linear in $|\mathcal{T}|$, using adapted data structures.

4.4 Algorithms and complexity

Our main algorithms are given in Fig. 2, page 9. As explain before, one must first compute the values for α_k, γ and the binomial coefficients that are needed in the process. In particular, γ having three parameters that can all three be



RandomNumberOfSecondarySources(n,s)

```
1 if n = s then
2     return 0
3 d ←Random( [α_k(n, s)] )
4 u ← 0
5 while d > 0 do
6     u ← u + 1
7     d ← d − C(n, s) · γ(ks, u, n − s − u) · α_k(n − s, u)
8 return u
```

RandomlySetTransitionsFromSources($\mathcal{T}$,$\mathcal{U}$,$\mathcal{R}$,δ)

```
  // Transitions are added to δ during the process
1 if T = ∅ then
2     return
3 (p, a) ←Remove the first element of T
4 if Random( [γ(|T|, |U|, |R|)] ) ≤ u · γ(|T| − 1, |U| − 1, |R|)
  then
5     q ←Remove the first element of U
6     δ(p, a) ← q
7 else
8     q ← Random(U ∪ R ∪ {⊥})
9     if q ≠ ⊥ then
10        δ(p, a) = q
11 RandomlySetTransitionsFromSources(T,U,R,δ)
```

RandomAcyclicAutomaton(n,s,δ)

```
1 if n = s then
2     δ =empty function
3     return
4 u ← RandomNumberOfSecondarySources(n,s)
5 B ←RandomAcyclicAutomaton(n − s,u,δ)
6 T ← ∅
7 for p ∈ {n − s + 1, . . . , n} and a ∈ A do
8     Add (p, a) in T
9 U ← {n − s − u + 1 . . . , n − s}
10 R ← [n − s − u]
11 RandomlySetTransitionsFromSources(T,U,R,δ)
```

Fig. 2. On the left a random acyclic automaton with 30 states on a two-letter alphabet. On the right our main algorithms. The states are labelled in a specific way during the process, but this does not change the uniformity of the unlabelled result, since we follow the correct counting numbers for the sources, secondary sources and transitions between them.

proportional to n, there are $\Theta(n^3)$ numbers to store. Thanks to Eq. (5), each new value of γ is computed in a constant number of arithmetic operations, giving the following result.

Theorem 1 (Preprocessing). *The preprocessing step of the algorithm, where all the possibly needed values of α_k, γ and $\binom{n}{s}$ are computed can be done using $\mathcal{O}(n^3)$ arithmetic operations and the memory to store $\mathcal{O}(n^3)$ numbers.*

Once the preprocessing is done, the random generation can be performed efficiently, as stated in the following theorem.

Theorem 2 (Generation). *After the preprocessing, the random generation of an acyclic automaton with n state can be done in a linear number of arithmetic operations and random generations of integers.*

Notice that, as it is usually the case when using the recursive method, the numbers involved in the computations are huge. This is why our theorems are stated in terms of number of arithmetic operations: one cannot consider that such operations can be done in constant time in real implementations. Alternative algorithms for the recursive method that use floating point arithmetic have been studied, but this is beyond the scope of this article.

4.5 A lazy strategy

A common strategy for that kind of algorithms is the lazy strategy, which consists in computing the values for α_k and γ only when needed. They are still stored, but the computations are done on the fly. This strategy proves to be very efficient in practice in our case, because of the specific shape of a uniform random acyclic automata (as depicted in Fig. 3).

If $\mathcal{A}$ is an acyclic automaton, let sources($\mathcal{A}$) be its number of sources and let pruned($\mathcal{A}$) be the acyclic automaton obtained when removing the sources and their outgoing transitions. We define the *width* width($\mathcal{A}$) of an acyclic automaton $\mathcal{A}$ by width($\mathcal{A}$) = max {sources($\mathcal{A}$), width(pruned($\mathcal{A}$))} if $\mathcal{A}$ has at least one state and width($\mathcal{A}$) = 0 otherwise. The width of an acyclic automaton $\mathcal{A}$ is therefore the maximum size of a layer of sources obtained when repeatedly pruning $\mathcal{A}$.

We aim at using the width of the output automaton as a parameter for the complexity of our algorithm. The main motivation for this is the typical flat shape of a random acyclic automaton (Fig. 3). Assume that the algorithm produces an automaton $\mathcal{A}$ of width w. Then the various values taken by u in the algorithms are always smaller than or equal to w, and those taken by $|\mathcal{T}|$ smaller than or equal to $k \cdot w$. Therefore, the lazy strategy only computes values for $\gamma(t, u, r)$ for $t \leq k \cdot w$, $u \leq w$ and $r \leq n$, which requires $\mathcal{O}(n \cdot w^2)$ time and space. However, the idea would not work without a shortcut to compute the values of $\alpha_k(n, s)$, which is needed in the algorithms; indeed, the variable u in Eq. (4) takes values that are bigger than w, especially considering the inductive nature of this equation. Fortunately, we can use the inclusion-exclusion formula of Eq. (2) instead, which can be computed using $\mathcal{O}(n^2 \cdot w)$ arithmetic operations according to Lemma 1. This gives the following result.

Fig. 3. The shape of a random acyclic automata with 200 states on a two-letter alphabet. The initial state is on the left, and we have represented the number of sources seen at each step when repeatedly pruning the automaton. The width of this automaton is 6, and its shape is typical of what is observed under the uniform distribution.

Theorem 3 (Lazy Strategy). *Using the lazy strategy in the random generator, the generation algorithm (including the computation of the required values) needs* $\mathcal{O}(n^2 \cdot w)$ *arithmetic operations, where* w *is the width of the generated acyclic automaton.*

Theorem 3 is not enough to prove the generic efficiency of the lazy strategy: one would also needs to show that, with high probably, a random uniform acyclic automaton with n states has a small width with respect to n. Experimentation indicates that it should be true, and trying to prove this is ongoing work. However, there is no reason to avoid this strategy, which is at least as good as the classical one even when the generated acyclic automaton has a large width.

5 Conclusion and Experiments

Using some optimizations on an inductive decomposition of acyclic automata, we proposed a random generator that is efficient enough to make experimental statistics on automata of size up to 1000 or a bit more. We relied on the alternative description of β_k by γ to lower the initial complexity, and used inclusion-exclusion formulas to make the lazy strategy possible.

We have implemented our random generator in the interpreted language Python, which is clearly not the best choice for computational speed. However, we could easily generate accessible automata of size 1000 in a reasonable amount of time (a few minutes to generate 100 automata on a personal computer).

In our experiments, we considered that each state is final with probability $\frac{1}{2}$. In our first test, we computed experimentally the probability that a random acyclic automaton with final states is minimal. Notice that minimality is easier to check for acyclic automata than for general automata (see [2] for the details). The results for alphabets of size 2 and 3 are depicted in Fig. 4.

We also computed the number of words in the finite language recognized by a random acyclic automaton with final states. Since it grows very fast, we switch to logarithms by calculating logarithm of the geometric mean of the number of recognized words. It seems to indicate that random acyclic automata are a very compact way to describe huge random sets of words (see Fig. 5).

Acknowledgments: we would like to thanks Arnaud Carayol for the very fruitful discussions we had during the preparation of this article.

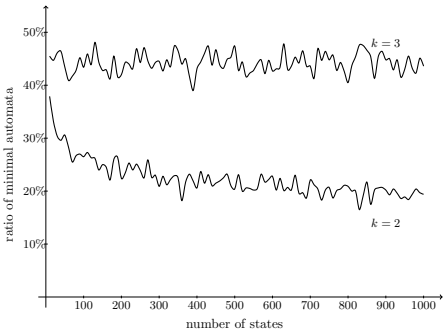
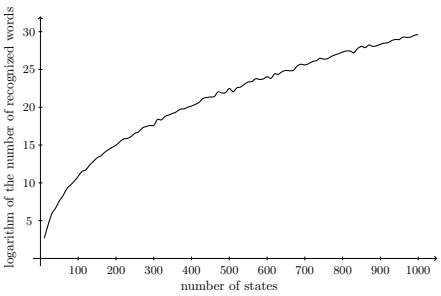


Fig. 4. The ratio of minimal automata for alphabet of size 2 and 3. Each curve has been obtained by generating 1000 acyclic automata for 101 different sizes, from 10 to 1000. Note that one can significantly increase the ratio by forcing states with no outgoing transition to be final (hoping that there is only one such state, which is often the case experimentally).

Fig. 5. The natural logarithm of the number of recognized words for an alphabet of size 2. The curve has been obtained by generating 1000 acyclic automata for 101 different sizes, from 10 to 1000. It is difficult to guess the function behind that kind of experimental curves, but subexponential growth like $x \mapsto e^{\sqrt{x}}$ is a possibility.



References

1. A. Akhavi, I. Klimann, S. Lombardy, J. Mairesse, and M. Picantin. On the finiteness problem for automaton (semi)groups. *IJAC*, 22(6), 2012.
2. M. Almeida, N. Moreira, and R. Reis. Exact generation of minimal acyclic deterministic finite automata. *Int. J. Found. Comput. Sci.*, 19(4):751–765, 2008.
3. A. S. Antonenko. On transition function of mealy automata with finite growth. *Matematychni Studii*, 29(1), 2008.
4. V. Carnino and S. De Felice. Sampling different kinds of acyclic automata using Markov chains. *TCS*, 450:31–42, 2012.
5. M. Domaratzki. Improved bounds on the number of automata accepting finite languages. In *DLT'02, LNCS 2450*, pages 209–219, 2002.
6. M. Domaratzki, D. Kisman, and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002.
7. P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge Univ. Pr., 2009.
8. P. Flajolet, P. Zimmermann, and B. V. Cutsem. A calculus for the random generation of labelled combinatorial structures. *TCS*, 132(2):1–35, 1994.
9. V. A. Liskovets. Exact enumeration of acyclic deterministic automata. *Discrete Applied Mathematics*, 154(3):537–551, 2006.
10. A. Nijenhuis and H. Wilf. *Combinatorial algorithms*. Computer science and applied mathematics. Academic Press, 1975.
11. R. P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge Univ. Pr., 2000.