UNIVERSITÉ PARIS-EST

THÈSE DE DOCTORAT DE l'UNIVERSITÉ PARIS-EST

Spécialité : **Informatique**

École doctorale Mathématiques et Sciences et Technologies de l'Information et de la Communication (MSTIC)

Présentée par

Sven DE FÉLICE

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PARIS-EST

Automates codéterministes et automates acycliques : analyse d'algorithme et génération aléatoire

---

Soutenue le **1er juillet 2014**, devant le jury composé de :

Mme **Frédérique Bassino**	Univ. Paris-Nord	Examinatrice
M. **Julien Clément**	Univ. de Caen	Examinateur
M. **Philippe Duchon**	Univ. de Bordeaux	Examinateur
M. **Pierre-Cyrille Héam**	Univ. de Franche-Comté	Examinateur
Mme **Ines Klimann**	Univ. Paris-Diderot	Examinatrice
M. **Cyril Nicaud**	Univ. Paris-Est	Directeur de thèse
M. **Antonio Restivo**	Univ. de Palerme	Rapporteur
Mme **Michèle Soria**	Univ. Pierre et Marie Curie	Rapporteuse

Remerciements

La personne qui s'apprête à lire ce rapport aurait tord de croire que les résultats qu'il présente sont dus au seul travail de son auteur. En espérant que ces travaux seront utiles à la communauté scientifique même de la façon la plus humble, l'auteur, qui écrit ces lignes, tient à préciser que, même si il en a rédigé le compte-rendu, il les doit aussi aux idées, aux soutiens et aux conseils d'un grand nombre de personnes qui, finalement, n'auront reçu en contrepartie qu'une rétribution presque inexistante comparée à celle que s'apprête à recevoir celui qui rédige.

C'est pour cela que je tiens à énumérer pour les remercier les personnes qui ont contribué de près ou de loin ou de façon indirecte, à la rédaction de ce rapport de thèse.

Je remercie ma famille et mes proches qui, peut-être même sans s'en rendre compte, m'ont soutenu le long de ces années de doctorat.

Je remercie Sylvie Cach, Claire David, Laura Giambruno, Carine Pivoteau, Viviane Pons, Omar Aït Mous, Arnaud Carayol, Vincent Carnino, Julien David, Samuele Giraudo, Patrice Hérault, Trong Hieu Dinh, Sylvain Lombardy, Rémy Maurice, Matthieu Picantin, Dominique Revuz Andrea Sportiello et Marc Zipstein pour m'avoir fait bon accueil, pour m'avoir donné de bons conseils et de bonnes idées et pour avoir été des collègues agréables et sur qui j'ai pu compter.

Je remercie aussi Frédérique Bassino, Inès Klimann, Julien Clément, Philippe Duchon et Pierre-Cyrille Héam d'avoir acceptés de faire partie du jury de ma soutenance.

Je remercie Michèle Soria d'avoir accepté de rapporter ma thèse et surtout de l'avoir fait si rapidement. Je remercie Antonio Restivo d'avoir accepté de rapporter ma thèse.

Et surtout, et enfin, je remercie la personne sans qui je n'aurais absolument rien pu faire, que j'ai placé en dernière position de cette liste afin de faire ressortir l'importance de sa contribution, qui a su aiguiller mes choix et diriger mon travail en montrant toujours une grande compréhension et une inaltérable patience, qui m'a aussi rendu mon courage et mon énergie lorsque parfois ceux-là venaient à me manquer, mon directeur de thèse Cyril Nicaud.

Table des matières

Introduction **5**

1 Notations et définitions **9**
1.1 Définitions et notations générales . . . 9
1.1.1 Notations . . . 9
1.1.2 Applications . . . 9
1.1.3 Permutations . . . 10
1.1.4 Asymptotique . . . 10
1.1.5 Probabilités . . . 10
1.1.6 Classes combinatoires . . . 12
1.1.7 Complexité . . . 12
1.1.8 Générateur Aléatoire . . . 14
1.2 Automates . . . 14
1.2.1 Langage . . . 14
1.2.2 Automate . . . 15
1.2.3 Langage reconnu . . . 16
1.2.4 Déterminisme . . . 18
1.2.5 Isomorphisme . . . 19
1.2.6 Déterminisation . . . 21
1.2.7 Minimisation . . . 22

2 Analyse de l'algorithme de Brzozowski **25**
2.1 Motivations . . . 25
2.2 Algorithme de Brzozowski . . . 28
2.3 Énoncé du théorème . . . 31
2.3.1 Idée de la preuve . . . 32
2.4 Preuves . . . 34
2.5 Perspectives sur l'algorithme de Brzozowski . . . 46

Automates acycliques **49**

3 Générateur aléatoire par chaîne de Markov **51**
3.1 Algorithme . . . 51
3.1.1 Chaîne de Markov . . . 52
3.1.2 Algorithme . . . 56
3.2 Preuves . . . 60
3.3 Algorithme sur automates acycliques minimaux . . . 63
3.4 Preuves . . . 66

3.5 Conclusion 73

4 Générateur aléatoire par la méthode récursive 75
4.1 Algorithme général 75
4.1.1 Décomposition d'automates acycliques 75
4.1.2 Algorithme 82
4.2 Algorithme optimisé 86
4.2.1 Calcul de β 86
4.2.2 Stratégie paresseuse 87
4.3 Algorithme final 91

Index 100

Bibliographie 103

Introduction

Cette thèse a pour domaine l'analyse d'algorithmes et la génération aléatoire. Pour analyser la performance des algorithmes on considère souvent la complexité dans le pire des cas, ce qui permet de garantir des bornes sur le temps d'exécution ou sur l'espace mémoire consommé.

Il existe cependant de nombreuses situations où un algorithme se comporte bien mieux « en pratique » que son pire des cas ne l'indique : il suffit pour cela que les cas « les pires » soient suffisamment « rares ». Pour quantifier mathématiquement ce type de propriétés, on analyse les performance d'un algorithme *en moyenne* ou *génériquement*.

L'exemple le plus célèbre de ce genre de comportements est probablement l'algorithme du tri rapide — QuickSort [Hoa62] —, dont la complexité dans le pire des cas est quadratique, alors que sa complexité en moyenne est en $O(n \log n)$. De fait, cet algorithme ou ses variantes sont utilisées dans les bibliothèques standards de nombreux langages de programmation.

Pour faire de l'analyse en moyenne d'algorithme il faut commencer par définir ce qu'est une entrée aléatoire, c'est-à-dire définir des distributions de probabilités sur les entrées disponibles. La façon la plus immédiate de le faire, et c'est celle qui sera majoritairement adoptée dans cette thèse, consiste à considérer la *distribution uniforme* où tous les objets de même « taille » ont une même probabilité d'être engendrés.

Utiliser la distribution uniforme sur les entrées de taille n reporte le problème sur comment compter le nombre total de ces entrées, puisque la probabilité d'un objet de taille n est l'inverse du nombre d'objets de cette taille. Nous aurons donc deux façons d'aborder les questions qui se posent : les probabilités discrètes et la combinatoire. Nous utiliserons les deux dans tout le manuscrit.

Notons que nous serons plus particulièrement intéressés par des résultats asymptotiques, comme montrer que la probabilité qu'une certaine propriété soit satisfaite tend vers 0 ou 1, et qu'on s'appuiera donc d'avantage sur les techniques de *combinatoire analytique* [FS09], qui fournissent les résultats pour faire de l'asymptotique, que sur d'autres champs de la combinatoire.

La théorie des automates et des langages rationnels existent depuis le tout début de l'informatique théorique. C'est un des domaines fondamentaux qui fait partie du socle commun de tous les informaticiens. Un mot est une suite de symboles et un langage est un ensemble de mots. Les suites de symboles interviennent naturellement en informatique — tout est codé par une suite de 0 et de 1 —, et donc les langages également : si on considère l'ensemble des représentations binaires des nombres premiers on obtient un langage, sur lequel on peut se poser des questions de nature informatique, comme par exemple d'écrire un programme pour décider si une séquence de 0 et de 1 donnée est dans ce langage de taille infini.

L'ensemble des langages rationnels forme une sous-famille de l'ensemble de tous les langages, qui possède de nombreuses bonnes propriétés mathématiques et algorithmiques. On peut notamment les représenter par des automates finis, qui sont des machines abstraites qui servent à décider facilement si un mot est dans le langage ou non. De nombreux algorithmes sur ces automates se sont développés depuis la fin des années 50, et ce sont ces objets et ces algorithmes qui vont être au centre de notre étude.

Cependant, il y a relativement peu de résultats sur le comportement en moyenne des automates finis et des langages rationnels. Il y a d'assez anciens résultats de dénombrement [Kor78], mais le vrai début d'une analyse d'algorithmes date des années 2000, avec des travaux sur la génération aléatoire [CP05, BN07], l'étude des automates unaires [Nic99], l'analyse de l'algorithme de Moore [BDN09, BDN12, Dav12, Dav10], le dénombrement des automates minimaux [BDS12] ou l'étude de la taille de la partie accessible avec application à la génération aléatoire [CN12].

Dans cette thèse, nous contribuons aux avancées de ce domaine en analysant l'algorithme de minimisation de Brzozowski et en nous intéressant à la génération aléatoire d'automates acycliques.

Cette thèse s'articule en trois chapitres originaux, un dédié à l'analyse de l'algorithme de Brzozowski et les deux autres à la génération aléatoire d'automates acycliques.

L'**algorithme de minimisation de Brzozowski** [Brz62] est un algorithme élégant qui permet de minimiser un automate, c'est-à-dire de trouver un automate — déterministe — le plus petit possible qui reconnaisse un langage donné. Cet algorithme a une mauvaise complexité dans le pire des cas, mais est réputé pour bien se comporter en pratique — cette réputation s'étend jusque sur la page qui lui est dédiée sur wikipedia —. Nous montrons que la complexité de l'algorithme est super-polynomiale pour la majeure partie des automates déterministes : pour la distribution uniforme, l'algorithme n'est donc pas compétitif avec les autres solutions polynomiales comme l'algorithme de Hopcroft [Hop71]. Ce résultat a fait l'objet d'une publication avec Cyril Nicaud [FN13a] et une extension est en cours de soumission.

Les **automates acycliques** sont des automates qui reconnaissent les ensembles finis de mots. Pour cette raison, ils apparaissent naturellement dans divers domaines d'application comme le traitement automatique des langues. Nous nous sommes intéressés à la génération aléatoire de ces objets, en respectant la distribution uniforme pour un nombre d'états fixé. Avoir un générateur aléatoire est un outil pratique à la fois pour tester empiriquement des algorithmes et pour guider le chercheur dans son étude des grands objets aléatoires. Nous proposons dans cette thèse deux solutions radicalement différentes pour mener à bien la génération : l'utilisation d'une **chaîne de Markov** et l'application optimisée de la **méthode récursive**. C'est surtout cette dernière méthode qui s'avère être efficace, quoique moins souple, et qui nous a permis d'observer par exemple qu'une proportion non négligeable d'automates acycliques sont minimaux. Ces méthodes ont fait l'objet

de publications avec Vincent Carnino pour les chaînes de Markov [CF11, CF12] et avec Cyril Nicaud pour la méthode récursive [FN13b].

Chapitre 1

Notations et définitions

1.1 Définitions et notations générales

Ce chapitre liste les notations générales et les termes qui seront utilisés tout au long de ce manuscrit. Des notations plus spécifiques apparaissent également au cours des chapitres qui suivent.

Dans tout le manuscrit, chaque nouveau terme qui ressort de son contexte de *cette* façon *ou alors qui ressort de* cette *façon de son contexte* est référencé dans un index à la fin de l'ouvrage.

1.1.1 Notations

On note $\mathbb{N}$ l'ensemble des entiers naturels. On dit qu'un ensemble est *dénombrable* s'il est en bijection avec un sous-ensemble de $\mathbb{N}$. Pour un entier naturel n non nul, on note $[n]$ l'ensemble des entiers i tel que $1 \leq i \leq n$. On note $n!$ la n-ième valeur de la suite factorielle. Par convention on pose $0! = 1$.

Pour deux réels a et b, $a \leq b$ on note $[a, b]$ l'intervalle des nombres réels compris entre a et b, a, b inclus.

Pour un ensemble E on note $\mathscr{P}(E)$ l'ensemble de ses parties. C'est à dire $a \in \mathscr{P}(E)$ si $a \subset E$. Si E est fini on note $|E|$ son cardinal.

L'ensemble vide est noté $\emptyset$.

Pour un réel positif non nul x, $\log(x)$ représente la valeur logarithmique en base e de x et $\exp(x)$ sa valeur exponentielle.

Pour deux entiers a et b on note $a|b$ pour dire que l'entier a divise l'entier b.

1.1.2 Applications

Soit f une application allant d'un ensemble E vers un ensemble F. Pour un sous-ensemble H de E on note $f(H)$ l'ensemble des éléments de F qui possèdent un antécédent dans H. Pour un sous-ensemble G de F on note $f^{-1}(G)$ l'ensemble des éléments de E qui ont pour image un élément de G.

Les notations suivantes sont utilisées lorsque $F = \mathbb{N}$. Soit b un entier naturel, alors on note :

— $\{f = b\}$ l'ensemble des éléments e de E qui vérifient $f(e) = b$.

— $\{f \leq b\}$ l'ensemble des éléments e de E qui vérifient $f(e) \leq b$.

Si f est injective, pour un élément i appartenant à F on note $f^{-1}(i)$ l'unique antécédent de f.

Si $E = F$, la *taille de l'application* f est définie comme le cardinal de E.

Pour tout élément e de E on note $f^i(e)_{i\in\mathbb{N}}$ la suite définie par la récurrence suivante $f^0(e) = e$ et pour tout entier naturel i non nul $f^i(e) = f^{i-1}(e)$. L'ensemble de toutes les valeurs prises par la suite $f^i(e)$ est appelé l'*orbite* de e par l'application f ou l'orbite de e s'il n'y a pas d'ambigüité. La taille ou la longueur d'une orbite est son cardinal. Lorsque tout élément de l'orbite possède un antécédent dans l'orbite on appelle cette orbite un *cycle*.

Pour tout entier naturel n non nul on note $\mathfrak{F}_n$ l'ensemble des applications de $[n]$ dans $[n]$, il y en a n^n.

1.1.3 Permutations

Pour tout entier naturel n non nul, on note $\mathfrak{G}_n$ l'ensemble des applications bijectives de $[n]$ dans $[n]$, autrement dit, $\mathfrak{G}_n$ est l'ensemble des *permutations* de taille n. On a $|\mathfrak{G}_n| = n!$.

On appelle *cycle*, une permutation qui possède une et une seule orbite de longueur plus grande que 1. Toute permutation peut être vue comme une composition de un ou plusieurs cycles. De plus, les termes d'une telle composition commutent et la composition est unique à l'ordre près.

1.1.4 Asymptotique

Soient a_n et b_n deux suites de réels qui ne s'annulent plus au bout d'un certain rang. On dit que a_n et b_n sont équivalentes si la suite $\frac{a_n}{b_n}$ tend vers 1. C'est une relation d'équivalence sur les suites. On exprime cette relation par la notation $a_n \sim b_n$. De plus pour toute suite c_n, si $a_n \sim b_n$ alors $c_n a_n \sim c_n b_n$. On dit que b_n *domine* a_n si il existe un entier M tel que $|a_n| \leq M|b_n|$ à partir d'un certain rang et on note $a_n = O(b_n)$ ou $b_n = \Omega(a_n)$. La relation de domination est une relation d'ordre large. Si a_n domine et est également dominée par b_n alors on note $a_n = \Theta(b_n)$.

On peut remarquer que si a_n et b_n sont deux suites équivalentes alors a_n est dominée par b_n — et b_n est dominée par a_n —.

On dit qu'une suite en n, est *super-polynomiale* si elle domine n'importe quel polynôme en n.

1.1.5 Probabilités

La partie qui suit rappelle plusieurs notions de probabilités élémentaires qui vont être utilisées. Pour plus de précisions dans ce domaine, le lecteur peut se référer à [Fel68].

Soit Ω un ensemble fini ou dénombrable. Une *distribution* p sur l'ensemble Ω est une application définie sur $\mathscr{P}(\Omega)$ et à valeur dans $[0, 1]$ qui vérifie les propriétés suivantes.

— Pour tout couple A,B de sous-ensemble de Ω tels que $A \cap B = \emptyset$ alors $p(A \cup B) = p(A) + p(B)$.

— Pour toute famille dénombrable $(A_i)_{i\in\mathbb{N}}$ de sous-ensemble de Ω alors

$$p\left(\bigcup_{n\in\mathbb{N}} A_n\right) \leq \lim_{n\to\infty} \sum_{i\leq n} p(A_i).$$

— $p(\Omega) = 1$.

Dans ce contexte, les éléments de $\mathscr{P}(\Omega)$ sont appelés des *évènements*, les valeurs prises par p sont appelées *probabilités*. Lorsqu'il n'y a pas d'ambigüité sur la distribution considérée on note $\mathbb{P}(A)$ la probabilité associée à l'évènement A.

Si Ω est fini de cardinal n, alors on définit la *distribution uniforme* sur E comme la distribution qui associe à tout sous-ensemble A de E la probabilité $\mathbb{P}(A) = \frac{|A|}{n}$.

Par exemple, la distribution uniforme sur $\mathfrak{G}_n$ associe le nombre $\mathbb{P}(\{\sigma\}) = \frac{1}{n!}$ à chaque permutation σ de $\mathfrak{G}_n$. La distribution uniforme sur $\mathfrak{F}_n$ associe à chaque application f de $\mathfrak{F}_n$ le nombre $\mathbb{P}(\{f\}) = \frac{1}{n^n}$.

On a la propriété suivante, si $(A_n)_{n\in\mathbb{N}}$ est une famille dénombrable d'évènement tous deux à deux disjoints alors on a l'égalité $\mathbb{P}(\cup_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} \mathbb{P}(A_i)$.

Soit Ω un ensemble, $\mathbb{P}_\Omega$ une distribution sur Ω et A un évènement tel que $\mathbb{P}(A) \neq 0$. L'application f de $\mathscr{P}(A)$ dans $[0,1]$, qui à chaque évènement C, $C \subset A$, associe le nombre $\frac{\mathbb{P}(C)}{\mathbb{P}(A)}$ est une distribution sur l'ensemble A. On la note $\mathbb{P}(\cdot \mid A)$ et on appelle la valeur $\mathbb{P}(C \mid A)$ la probabilité de l'évènement C *sachant* A.

Soient A et B deux évènements. On dit que A et B sont deux évènements indépendants si $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$.

Soit Ω un ensemble et soit un couple (X, f) où X est une application d'un ensemble Ω dans un ensemble F et f une distribution sur Ω.

Le couple (X, f) est appelé une *variable aléatoire*. Souvent, on ne nomme que X en précisant la distribution à laquelle elle est associée. Dans ce contexte, pour tout élément e de F, on note $\mathbb{P}(X = e)$ la probabilité $f(\{X = e\})$.

Pour illustrer cette définition, on peut parler de la variable aléatoire X qui compte le nombre de cycles d'une permutation avec la distribution uniforme sur les permutations de taille 4. Ici $\mathbb{N}$ joue le rôle de F, $\mathfrak{G}_4$ joue le rôle de Ω et pour toute permutation σ de taille 4, $f(\{\sigma\}) = \frac{1}{24}$. On a $\mathbb{P}(X = 1) = \frac{6}{24}$, $\mathbb{P}(X = 2) = \frac{3+8}{24}$, $\mathbb{P}(X = 3) = \frac{6}{24}$, $\mathbb{P}(X = 4) = \frac{1}{24}$, et pour toutes autres valeurs i, $i \notin \{1,2,3,4\}$, on a $\mathbb{P}(X = i) = 0$.

Dans tous les cas, l'application X définit une partition de l'ensemble Ω ce qui signifie que l'on a $\sum_{e\in F} \mathbb{P}(X = e) = 1$.

Soit (X, f) une variable aléatoire sur Ω, avec X allant de Ω dans F et soit Y une application allant de F dans G, alors la composition de l'application X suivit de l'application Y est une variable aléatoire qui prend ses valeurs dans G.

Soit (X, f) une variable aléatoire où X prend ses valeurs dans l'ensemble des entiers naturels $\mathbb{N}$. On appelle l'espérance de la variable X et on note $E(X)$ la valeur $\sum_{i\in\mathbb{N}} i\mathbb{P}(X = i)$. Nous faisons remarquer que l'espérance peut être infinie. Lorsque $E(X)$ est finie, la valeur $E((X - E(X))^2)$ est appelée la *variance* de X. L'*écart type* d'une variable aléatoire est la racine carrée de sa variance.

Si (X, f) est une variable aléatoire d'espérance finie μ et d'écart type σ alors on a la relation bien connue de Bienaymé-Tchebychev

$$\mathbb{P}(\alpha \leq |X - \mu|) \leq \frac{\sigma^2}{\alpha^2}.$$

1.1.6 Classes combinatoires

Une classe *combinatoire* est un couple $(\mathcal{C}, f)$ où $\mathcal{C}$ est un ensemble et f une application allant de $\mathcal{C}$ vers $\mathbb{N}$ qui vérifie que pour tout entier naturel n l'ensemble $\{f = n\}$ est fini.

Pour un élément c de $\mathcal{C}$ on appelle $f(c)$ sa *taille* , on la note $|c|$ et on note $\mathcal{C}_n$ l'ensemble des éléments de taille n.

Soit $\mathcal{C}$ une classe combinatoire et soit une distribution sur $\mathcal{C}$. On dit qu'un sous-ensemble E de $\mathcal{C}$ est un ensemble *générique* si la suite $\mathbb{P}(E \mid \mathcal{C}_n)$ tend vers 1 lorsque n tend vers l'infini. Souvent on prend sur $\mathcal{C}$ une distribution qui est uniforme sur les $\mathcal{C}_n$. Une propriété sur les éléments de $\mathcal{C}$ est dite générique si l'ensemble des éléments qui la vérifie est générique. Le complémentaire d'un ensemble générique sur $\mathcal{C}$ est appelé un ensemble *négligeable*.

Il en découle la caractérisation suivante : un ensemble F est négligeable si et seulement si $\mathbb{P}(F \mid \mathcal{C}_n)$ tend vers 0 lorsque n tend vers l'infini.

On en déduit que l'union de deux ensembles négligeables est négligeable et donc que l'intersection d'un nombre fini d'ensembles génériques est un ensemble générique.

Si le lecteur souhaite en savoir plus sur les classes combinatoires et connaître des techniques de calcul avec ces classes il peut consulter [FS09].

1.1.7 Complexité

Un algorithme est un assemblage de petites opérations que l'on pourrait appeler *opérations élémentaires*. Par exemple l'addition de deux entiers, la comparaison de deux réels sont des opérations élémentaires. Pour chaque entrée de l'algorithme, le problème est résolu après l'exécution d'un certain nombre d'opérations élémentaires, par analogie on appelle ce nombre le *temps d'exécution* de l'algorithme associé à l'entrée en question.

On appelle la *complexité dans le pire des cas* d'un algorithme la fonction mathématique qui à un nombre entier n associe le maximum des temps d'exécutions pour toutes les entrées de taille n. C'est une application à valeurs entières, et qui, dans la grande majorité des cas, croît avec n, la taille des entrées.

La « taille » d'une entrée est délicate à définir. Elle peut être vue comme le nombre de lettres nécessaires pour décrire le problème. Cette définition est ambigüe car la taille d'une donnée dépend de plusieurs choses dont la taille de l'alphabet et de la façon de décrire le problème. De même, le temps d'exécution dépend de l'ensemble des opérations élémentaires que l'on utilise. En partie pour ces deux raisons, le calcul précis de la fonction de complexité d'un algorithme est fastidieux

et inutile. On préfère plutôt calculer un équivalent asymptotique de cette fonction ou en donner une majoration par une application connue et « simple ». Par exemple le tri d'un tableau de n d'objets comparables par l'algorithme de tri par sélection possède dans le pire des cas une complexité en $O(n^2)$.

Dans ce manuscrit nous décrivons nos algorithmes par des opérations élémentaires dites *arithmétiques*. Ce qui signifie que l'on considère que chaque opération « basique » sur les entiers, comme l'addition ou la division, est une opération qui utilise un nombre constant d'opérations élémentaires, donc indépendamment de la taille des entiers sur laquelle l'opération agit.

Ce modèle ne correspond pas exactement à ce qui se passe en pratique car le « véritable » temps d'exécution dépend aussi de la taille des entiers.

En plus de la complexité dans le pire des cas, il existe d'autres considérations sur le temps d'exécution. Dans ce manuscrit, nous parlerons aussi de *complexité en moyenne* et de complexité générique.

La complexité *en moyenne* est associée à une distribution sur l'ensemble des entrées de taille n. La valeur de la complexité en moyenne d'un algorithme est une application qui associe, une taille de donné n à l'espérance du temps d'exécution de l'algorithme selon la distribution de l'ensemble des entrées de taille n.

Une *complexité générique* est également associée à une distribution sur l'ensemble des entrées de l'algorithme et correspond à la complexité d'un ensemble générique de ces entrées.

Formellement, si on représente l'ensemble des entrées de l'algorithme par une classe combinatoire $\mathcal{C}$, que l'on pose $\mathbb{P}_n$ la distribution sur $\mathcal{C}_n$ et f l'application allant de $\mathcal{C}$ vers $\mathbb{N}$ qui représente le temps d'exécution, alors h la complexité dans le pire des cas est définie par

$$h(n) = \max_{\omega \in \mathcal{C}_n} f(\omega),$$

et g la complexité en moyenne est définie par

$$g(n) = \sum_{\omega \in \mathcal{C}_n} f(w) \cdot \mathbb{P}_n(\{\omega\}).$$

Souvent la distribution utilisée est une distribution qui est uniforme sur des entrées de même taille.

Les différents types de complexité évoqués ici ne considèrent que le temps d'exécution.

On peut aussi s'intéresser à la complexité en espace ou à d'autres mesures de performance.

Si le lecteur souhaite approfondir ses connaissances sur l'analyse d'algorithme en général, il peut consulter [Knu73a],[Knu81] et [Knu73b] qui en sont les ouvrages fondateurs.

1.1.8 Générateur Aléatoire

Un algorithme de *génération aléatoire* — ou *générateur aléatoire* — produit des objets selon une certaine distribution en fonction d'un ou de plusieurs paramètres.

Lorsqu'on décrit un générateur, on ne se pose pas le problème de la production d'informations aléatoires. On suppose que l'on possède des générateurs aléatoires élémentaires « parfaits » qui produisent des structures simples selon des distributions connues, et on utilise ces générateurs pour engendrer des structures plus complexes. Un exemple de générateur aléatoire élémentaire est l'algorithme qui tire un nombre entier aléatoirement entre 1 et un paramètre n selon une distribution « uniforme ».

Dans ce manuscrit on suppose que l'exécution d'un générateur élémentaire prend un nombre constant d'opérations élémentaires, bien qu'en pratique ce temps dépend souvent du paramètre.

Les objets produits par un générateur sont des éléments d'une classe combinatoire et possède une taille. Dans ce manuscrit on parle uniquement de générateurs aléatoires où la distribution est uniforme sur des objets de même taille.

Conventionnellement, la mesure de la complexité d'un algorithme de génération ne suit pas la règle générale. Elle n'est pas donnée en fonction de la taille des entrées mais en fonction de la taille des objets à produire en sortie. Dans le manuscrit, la taille est toujours un paramètre des algorithmes décrits et elle est appelée n.

À titre d'exemple, on peut générer uniformément des permutations de taille n en temps linaire avec l'algorithme de Fischer-Yats aussi appelé mélange de Knuth-Shuffle — voir [Knu81] — ou encore des arbres binaires uniformes à n nœuds avec l'algorithme de Rémy — voir [Ré85] —. Des techniques plus générales permettent de traiter de nombreuses classes d'objets combinatoires, par exemple la méthode récursive — voir [NW75],[FZC94] —.

Plus récemment sont apparus les générateurs de Boltzmann [DFLS04]. Qui sont des procédés astucieux et très efficaces pour faire de la génération aléatoire.

1.2 Automates

Cette section termine le chapitre des notations et définitions, elle introduit des notions sur les automates. Pour plus de détails sur les automates et les langages le lecteur peut consulter le livre [HU79].

1.2.1 Langage

Soit A un ensemble fini. On définit un *mot* sur A comme une suite finie d'éléments de A.

Par exemple si $A = \{a, b\}$ alors a, ab, aaa, bbb sont des mots sur A. L'ensemble des mots de A est notée A^*. Dans ce contexte, les éléments de A sont appelés les *lettres* et A est l'*alphabet*. Dans la suite de tout ce document, sauf mention contraire,

l'alphabet possède au moins 2 lettres a et b et son cardinal est noté k.

La *longueur d'un mot* est le nombre de lettres qui le compose. Par exemple a est de longueur 1, bbb et aaa sont de longueur 3. Il existe un mot et un seul de longueur 0, on le nomme le mot vide et on le note ε. La longueur d'un mot $w \in A^*$ est noté $|w|$. Pour un mot w de longueur non nulle et pour un entier i, $1 \leq i < |w|$, on note w_i le i-ème terme de la suite w. Par exemple si $w = abbaabbab$, $w_1 = a$, $w_2 = b$ et $w_8 = a$.

Un *langage* sur un alphabet A est une partie de A^*. Par exemple $\{aa\}$, $\{a, b\}$, $\{\varepsilon\}$, $\emptyset$, $\{\varepsilon, a, aa, aaa, \dots\}$ sont des langages.

On peut classer les langages en plusieurs types, l'un d'eux est appelé l'ensemble des *langages rationnels*. Les langages rationnels sont « reconnaissables » par un type d'objet théorique que l'on appelle *automate*. La suite expose différents aspects des automates.

Un mot w est une *puissance* d'un mot u s'il peut s'écrire comme une concaténation de plusieurs fois le même mot u. On note $\underbrace{u \cdot u \cdot u \dots u}_{\ell \text{ fois}} = u^\ell$ où ℓ est l'entier comptant le nombre de u.

Un mot est dit *primitif* s'il ne peut pas être écrit comme une puissance d'un mot strictement plus petit. Par exemple les mots $abab$, $aaaa$, $ababbabab$ ne sont pas primitifs car ils peuvent s'écrire $ab \cdot ab$, $a \cdot a \cdot a \cdot a$, $abab \cdot abab$. Les mots aba, $bbaa$, sont des mots primitifs.

Pour un mot w on appelle le *facteur primitif* de w, le plus petit mot u tel que $w = u^\ell$. Le facteur primitif d'un mot non-vide est de longueur au moins 1. Le facteur primitif d'un mot primitif est le mot lui-même.

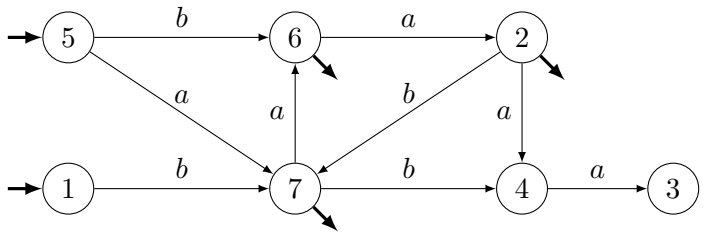
1.2.2 Automate

Un automate $\mathcal{A}$ est un n-uplet $\mathcal{A} = (Q, A, T, I, F)$ où Q est un ensemble fini, A est un alphabet, T une partie de l'ensemble $Q \times A \times Q$, I et F sont deux parties de Q. Les éléments de Q sont appelés les *états* de l'automate $\mathcal{A}$, A est appelé l'alphabet de $\mathcal{A}$, T est l'ensemble des *transitions* de l'automate $\mathcal{A}$, I sont les *états initiaux* de $\mathcal{A}$ et F les états *terminaux* de A. La *taille d'un automate* $\mathcal{A}$ que l'on note $|\mathcal{A}|$ est son nombre d'états, c'est à dire $|Q|$. L'ensemble T est aussi appelé la *structure de transition* de l'automate.

On peut représenter un automate comme un graphe orienté où les états de l'automate sont les sommets du graphe et les transitions de l'automate sont les arcs du graphe. Un arc allant d'un sommet q à un sommet p, étiqueté par une lettre c, correspond à une transition (q, c, p) — voir figure 1.1 —.

On appelle ce graphe le *graphe associé* à l'automate $\mathcal{A}$. Une transition (q, c, p) est notée $q \xrightarrow{c} p$.

On le verra dans la suite, la « nature » des états de l'automate n'est pas importante, ce sont les transitions d'un automate et la manière dont elle sont réparties les unes par rapport aux autres qui importe. À partir de maintenant, et sauf mention



$\mathcal{A} = (Q, A, T, I, F)$, $Q = [7]$, $A = \{a, b\}$,
$T = \{\ (1, b, 7), (2, a, 4),\ (2, b, 7),\ (4, a, 3),\ (5, a, 7),\ (5, b, 6),\ (6, a, 2),$
$(7, a, 6),\ (7, b, 4)\ \}$, $I = \{1, 5\}$, $F = \{2, 6, 7\}$.

FIGURE 1.1 – Une représentation de l'automate $\mathcal{A}$ sous la forme d'un graphe.

contraire, l'ensemble des états d'un automate de taille n est l'ensemble $[n]$, c'est à dire les entiers naturels compris entre 1 et n. Il arrive que l'on parle de l'étiquette d'un état au lieu de le désigner directement, par exemple on parle l'état d'étiquette 3 pour parler de l'état 3.

Un état qui ne possède aucune transition entrante est appelé un état *source* ou une source de l'automate.

1.2.3 Langage reconnu

Soit $\mathcal{A} = (Q, A, T, I, F)$ un automate. Un *chemin* dans l'automate $\mathcal{A}$ correspond à un chemin dans le graphe associé. Un chemin est une suite finie q_0, t_1, q_1, t_2,-$q_2, \ldots, q_d$, où les q_i sont des états de l'automate et les t_i sont des transitions telles que pour chaque entier naturel i, $1 \leq i \leq d$ on ait $t_i = q_{i-1} \xrightarrow{c_i} q_i$, c_i étant n'importe quelle lettre de A. La *longueur du chemin* est le nombre de transitions présentes dans le chemin. En l'occurrence c'est d.

On définit le *mot associé à un chemin* $C = q_0, t_1, q_1, t_2, q_2, \ldots, q_d$ comme la suite des lettres des transitions $t_2, t_3, \ldots, t_d$ écrites dans le même ordre que leur ordre dans le chemin. Le mot associé à C est $c_1 c_2 \ldots c_q$ où $t_i = q_{i-1} \xrightarrow{c_i} q_i$.

Un mot dans A^* est *reconnu* par un automate $\mathcal{A}$, $\mathcal{A} = (Q, A, T, I, F)$, s'il existe un chemin, dans l'automate, auquel le mot est associé et qui part d'un état initial et aboutit à un état terminal.

Il faut remarquer que pour un mot reconnaissable par un automate il peut exister plusieurs chemins qui lui sont associés dans cet automate. Pour un automate $\mathcal{A}$ donné on appelle le *langage reconnu* par l'automate l'ensemble des mots que reconnaît l'automate $\mathcal{A}$. On le note $\mathcal{L}(A)$.

Un *cycle d'un automate* — ou chemin cyclique — est un chemin de longueur au moins 1 qui ne passe pas deux fois par le même état et qui possède le même état au départ et à l'arrivée.

On remarque que un état d'un cycle est accessible si et seulement si tous ses états sont accessibles. Un cycle est dit *accessible* si le cycle contient un état accessible.

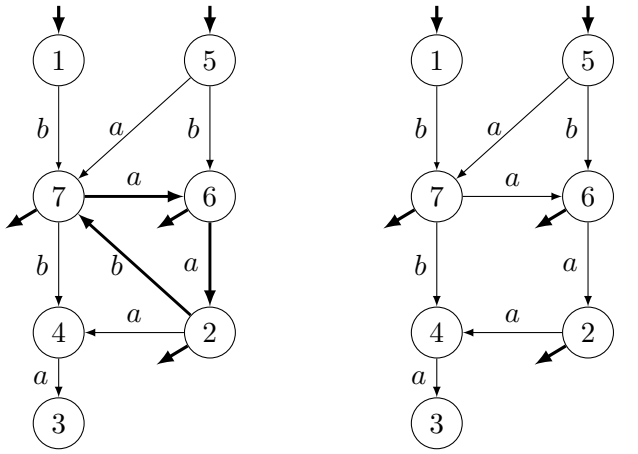


FIGURE 1.2 – Deux automates, l'un est acyclique l'autre ne l'est pas. Le mot *babaa* est reconnu par l'automate de gauche mais n'est par reconnu par celui de droite.

Pour une lettre a, on définit un *a-cycle* par une lettre comme un cycle qui emprunte uniquement des transitions étiquetées par la lettre a.

Soit un cycle — ou chemin cyclique —, on peut associer à ce cycle un mot à deux lettres de la façon suivante : on ne conserve que l'ordre des états parcourus et on remplace chaque état non terminal par une lettre et chaque état terminal par l'autre lettre. Le facteur primitif du mot trouvé est appelé la *période* du cycle. Si le mot trouvé est primitif alors cette période est égale à la longueur du cycle et on dit que le cycle est un *cycle primitif* — voir exemple figure 2.4 page 36 —.

On peut remarquer que dans un chemin cyclique, quelque soit l'état que l'on choisisse comme point de départ, la période du cycle reste la même, de même que ses propriétés d'être primitif et d'être accessible ne changent pas.

Un automate $\mathcal{A}$ est dit *acyclique* s'il ne possède pas de cycle — voir figure 1.2 —. C'est à dire que pour tout chemin $q_0, t_1, q_1, \ldots, q_\ell$ de $\mathcal{A}$, les états q_i, $i \leq \ell$ sont tous différents deux à deux. Dans un automate acyclique, tout chemin est de longueur bornée par le nombre d'états de l'automate. Donc la longueur des mots reconnus par l'automate est également bornée et on en déduit la même chose pour le nombre de mots reconnus par l'automate.

Soit un mot u sur un alphabet A. On appelle *miroir* du mot u le mot obtenu en renversant l'ordre de ses lettres. Si v est le miroir du mot u alors $|u| = |v|$ et pour tout entier naturel i, $1 \leq i \leq |v|$ on a $v_i = u_{|u|-i+1}$. Le miroir du miroir du mot w est w lui-même. Le miroir du mot vide est le mot vide.

Soit un langage L, le *langage miroir* de L est l'ensemble de ses mots miroirs. Un mot w appartient au miroir de L si le miroir de w appartient à L. Le miroir du miroir d'un langage L est L lui-même.

Le miroir d'un automate est l'automate construit à partir de $\mathcal{A}$ en inversant le sens de chaque transition et en permutant l'ensemble des états terminaux avec

l'ensemble des états initiaux. Si $\mathcal{A} = (Q, A, T, I, F)$ est un automate, alors $\mathcal{A}' = (Q, A, T', F, I)$ est son automate miroir où T' est défini ainsi : $T' = \{q \xrightarrow{c} p \in Q \times A \times Q \mid (p, c, q) \in T\}$.

Si un automate $\mathcal{A}$ reconnaît le langage L alors le miroir de $\mathcal{A}$ reconnaît exactement le miroir de L. Le miroir du miroir d'un automate $\mathcal{A}$ est $\mathcal{A}$.

Soit un automate $\mathcal{A}$. Un état p de $\mathcal{A}$ est dit *accessible* s'il existe un chemin allant d'un état initial de $\mathcal{A}$ et aboutissant à l'état p. Un automate dont tous les états sont accessibles est appelé un *automate accessible*. Un état q de $\mathcal{A}$ est dit *co-accessible* s'il existe un chemin allant de l'état q à un état terminal de $\mathcal{A}$. Un état co-accessible d'un automate $\mathcal{A}$ est accessible dans le miroir de $\mathcal{A}$. La *partie accessible* d'un automate est l'ensemble des états accessibles.

Un automate est dit *émondé* si tout ses états sont accessibles et co-accessibles. Autrement dit, un automate est émondé si pour tout état r de l'automate il existe un chemin qui part d'un état initial, qui passe par r et qui arrive à un état terminal.

On parle aussi de l'action d'*émonder* un automate lorsqu'on supprime de cet automate les états qui ne sont ni accessibles ni co-accessibles et les transitions reliées à ces états. Émonder un automate ne modifie pas le langage reconnu par cet automate.

1.2.4 Déterminisme

Un automate $\mathcal{A} = (Q, A, T, I, F)$ est dit *déterministe* s'il ne possède qu'un seul état initial — c'est à dire $|I| = 1$ — et si pour tout état q et toute lettre c il existe au plus qu'une seule transition t de l'automate qui part de q et qui a pour étiquette la lettre c. Pour un état q de Q on pose A_q l'ensemble des lettres des transitions sortant de q.

Dans le cas où de plus $|T| = |Q| \cdot |A|$, c'est à dire si pour tout état e et pour toute lettre c il existe une et une seule transition t telle que $t = q \xrightarrow{c} \cdot$ alors on dit de $\mathcal{A}$ que c'est un automate *déterministe complet*.

Si $\mathcal{A} = (Q, A, T, I, F)$ est déterministe complet on peut définir l'application δ allant de l'ensemble des couples $Q \times A$ dans Q de la façon suivante : pour tout couple (q, c) dans $Q \times A$ on associe le seul état p tel que $q \xrightarrow{c} p \in T$, donc $\delta(q, c) = p$. L'application δ est appelée la *fonction de transitions d'un automate*.

On peut compléter un automate déterministe qui n'est pas complet en ajoutant toutes les transitions qui ne sont pas définies et en les dirigeant vers un nouvel état appelé *état puits*. On complète l'automate en précisant que toutes les transitions sortant de l'état puits sont dirigées vers ce même état puits et que l'état puits n'est pas terminal. L'automate ainsi construit est complet, déterministe, et reconnaît le même langage que l'automate d'origine.

Un automate complet possède au moins un cycle ; en conséquence un automate acyclique ne peut être complet.

Il est quelquefois plus simple de considérer le *complété* d'un automate lors des preuves ou lors de la définition des algorithmes notamment lorsqu'il est question

d'automates acycliques. On désigne par $\perp$ l'état puits du complété d'un automate acyclique. On convient tout de suite que l'état puits ne participe pas à la taille de l'automate acyclique — le complété d'un automate acyclique de taille n possède « réellement » $n+1$ états —.

Si un automate est déterministe et sauf mention contraire, son état initial est étiqueté par 1. Si $\mathcal{A}$ est un automate déterministe complet on le note $\mathcal{A} = (Q, A, \delta, F)$ plutôt que $(Q, A, T, \{1\}, F)$. Si on veut préciser son état initial, on écrit $\mathcal{A} = (Q, A, \delta, i, F)$ où i est l'état initial.

Lorsque l'automate est complet on peut étendre l'application de δ aux mots de A^*. Pour un mot w et une lettre ℓ on pose :

$$\delta(q, \varepsilon) = q$$
$$\delta(q, \ell \cdot w) = \delta(\delta(q, \ell), w).$$

Dans un automate non-déterministe, l'ensemble des chemins, partant d'un même état et associé à un même mot w, peut être de cardinal 0, 1 ou plus. Dans un automate déterministe cet ensemble ne peut être que 0 ou 1.

Savoir si un mot est reconnu par un automate déterministe peut s'effectuer en $O(\ell)$ opérations où ℓ est la longueur du mot. Si l'automate n'est pas déterministe le nombre d'opérations peut être beaucoup plus grand.

1.2.5 Isomorphisme

Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates $\mathcal{A} = (Q_\mathcal{A}, A, T_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A})$, $\mathcal{B} = (Q_\mathcal{B}, A, T_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B})$. Nous dirons que $\mathcal{A}$ est un *sous-automate* de $\mathcal{B}$ si il existe une application injective i allant de $Q_\mathcal{A}$ dans $Q_\mathcal{B}$ et telle que

— pour tout état p, $p \in I_\mathcal{A}$, $i(p) \in I_\mathcal{B}$,

— pour tout état p, $p \in F_\mathcal{A}$, $i(p) \in F_\mathcal{B}$,

— pour toute transition $p \xrightarrow{c}, q$ de $T_\mathcal{A}$ alors $i(p) \xrightarrow{c} i(q)$ appartient à $T_\mathcal{B}$.

Intuitivement un sous-automate $\mathcal{A}$ est obtenu en supprimant des états et des transitions de $\mathcal{B}$.

Si $\mathcal{A}$ est un sous-automate de $\mathcal{B}$, par une injection i, cette injection n'est pas forcément unique. Lorsque l'injection i est explicitement donnée ou lorsque qu'elle est unique alors l'image de i est appelée l'ensemble des états *correspondant* aux états de $\mathcal{A}$, ce sont des états de $\mathcal{B}$.

Si l'injection est explicitement donnée on dit que $\mathcal{A}$ est un *sous-automate normalisé* de $\mathcal{B}$ et si l'ordre des étiquettes des états de $\mathcal{A}$ est le même que leurs états correspondant dans l'automate $\mathcal{B}$. Ce qui veut dire que pour tout couple d'états p, q, $p < q$ de l'automate $\mathcal{A}$ on a $i(p) < i(q)$.

On remarque que si il existe dans $\mathcal{A}$ un chemin allant d'un état p à un état q étiqueté par un mot w alors il existe dans $\mathcal{B}$ un chemin allant de l'état $i(p)$ vers l'état $i(q)$ et étiqueté par w.

De cette remarque on tire les trois propriétés suivantes. Si $\mathcal{A}$ est un sous-automate de $\mathcal{B}$ alors :

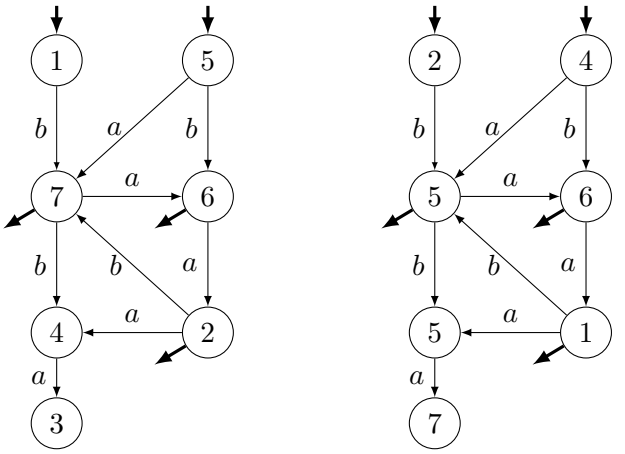


FIGURE 1.3 – Deux automates différents mais isomorphes.

— Si $\mathcal{B}$ est déterministe alors $\mathcal{A}$ est aussi déterministe.
— Si $\mathcal{B}$ est acyclique alors $\mathcal{A}$ est aussi acyclique.
— Si $\mathcal{A}$ est accessible et $\mathcal{B}$ est de même taille que $\mathcal{A}$ alors $\mathcal{B}$ est accessible.
— $\mathcal{L}(\mathcal{A}) \subset \mathcal{L}(\mathcal{B})$.

La relation « être un sous automate de » est une relation d'ordre large.

On dit qu'un automate $\mathcal{A}$ est *isomorphe* à un automate $\mathcal{B}$ si $\mathcal{A}$ est un sous-automate de $\mathcal{B}$ et $\mathcal{B}$ est un sous-automate de $\mathcal{A}$.

Si $\mathcal{A}$ et $\mathcal{B}$ sont isomorphes il existe un plongement i de $\mathcal{A}$ dans $\mathcal{B}$ qui en plus d'être injectif, est surjectif et engendre une bijection sur l'ensemble des état initiaux, sur l'ensemble des état terminaux et sur l'ensemble des transitions des deux automates.

Si $\mathcal{A}$ et $\mathcal{B}$ sont isomorphes alors :
— Ils ont la même taille.
— $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.
— $\mathcal{A}$ est acyclique si et seulement si $\mathcal{B}$ est acyclique.
— $\mathcal{A}$ est accessible si et seulement si $\mathcal{B}$ est accessible.

On pose $\mathbb{E}_n$ l'ensemble des automates de taille n. Parmi les automates $\mathbb{E}_n$ on peut considérer les classes d'équivalences d'automates isomorphes ou *classes d'isomorphismes*. On distingue les automates de $\mathbb{E}_n$ d'une part et les classes d'isomorphisme de $\mathbb{E}_n$ d'autre part. Les premiers forment l'ensemble des *automates étiquetés* et les secondes forment l'ensemble des *automates non étiquetés*. Dans la suite, si la catégorie d'un automate n'est pas mentionnée, on convient que c'est un automate étiqueté.

Tous les automates d'une même classe d'équivalence reconnaissent un même langage, ainsi, on définit le langage reconnu par un automate non étiqueté, comme étant le langage qui est reconnu par l'un de ses représentants étiquetés.

Soit $\mathcal{A} = (Q, T, I, F)$, un automate étiqueté de taille n. Pour connaître tous les automates étiquetés présents dans la classe d'isomorphisme de $\mathcal{A}$ il suffit de regarder les images des ensembles T, I, F par les permutations de $\mathfrak{G}_n$. Pour certains

automates, plusieurs permutations laissent invariants les ensembles T, I, F — par exemple si $T = I = F = \emptyset$ — autrement dit ces permutations laissent l'automate invariant. Dans d'autres cas, chaque paire de permutations différentes mêne à une paire d'automates différents. Pour cette raison, les classes d'isomorphisme de $\mathbb{E}_n$ n'ont pas toutes le même cardinal.

Lemme 1 *Soit $\mathcal{A}$, un automate déterministe acccessible de taille n. Alors le nombre d'automates étiquetés de la classe d'isomorphisme de $\mathcal{A}$ ne dépend que de n. Il est égal à $(n-1)!$ si on impose que 1 est toujours l'état initial.*

Preuve :

Soit $\mathcal{B} = (Q_\mathcal{B}, T_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B})$ un automate isomorphe à $\mathcal{A} = (Q_\mathcal{A}, T_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A})$ et soit σ un isomorphisme allant de $Q_\mathcal{A} = [n]$ vers $Q_\mathcal{B} = [n]$. L'isomorphisme est une permutation de $[n]$ dans $[n]$.

Montrons que si $\mathcal{A} = \mathcal{B}$ alors nécessairement σ est l'identité.

On suppose que $\mathcal{A} = \mathcal{B}$, on appelle i l'état initial. Soit p un état de $\mathcal{A}$. On pose C le chemin qui va de i à p et w le mot qui lui est associé. Un tel chemin existe car $\mathcal{A}$ est accessible. On pose C' l'image de ce chemin par σ. Le chemin C' va de $\sigma(i)$ à $\sigma(p)$. Le chemin C' est donc également associé au mot w puisque σ est un isomorphisme, or comme $\mathcal{A}$ est déterministe w étiquette un unique chemin donc $C = C'$, et donc $\sigma(p) = p$. □

Ce lemme nous permettra de simplifier grandement l'analyse et la génération aléatoire d'automates déterministes accessibles non étiquetés. Souvent le travail que l'on fait sur les structures combinatoires étiquetées est beaucoup plus facile que sur leur homologues non étiquetés. Dans le cas des automates déterministes accessibles d'une même taille — ou d'un sous ensemble de celui-ci — l'effort est le même puisque le rapport de leurs cardinaux est une valeur constante.

Par exemple pour tirer uniformément un automate accessible déterministe non étiqueté, il suffit de tirer un automate étiqueté et d'oublier ses étiquettes, la distribution reste uniforme puisque toutes les classes d'équivalences ont une même taille.

1.2.6 Déterminisation

La *déterminisation* d'un automate est un procédé qui permet, à partir d'un automate non déterministe, de construire un automate déterministe qui reconnaît le même langage que l'automate original.

Soit un automate $\mathcal{A} = (Q, A, T, I, F)$. On pose $\mathcal{D} = (\mathscr{P}(Q), A, \delta, I, \mathbf{F})$ un nouvel automate construit à partir de $\mathcal{A}$ et qui est déterministe complet. Les états de $\mathcal{D}$ sont les sous-ensembles des états de Q. L'application δ de $\mathscr{P}(Q) \times A$ dans $\mathscr{P}(Q)$ est définie de la façon suivante. Pour tout sous-ensemble d'état P (i.e. $P \subset Q$) et toute lettre c on pose $\delta(P, c) = \{q \in Q \mid \exists p \in P, p \xrightarrow{c} q \in T\}$. L'état initial de $\mathcal{D}$ est I, l'ensemble des états initiaux de $\mathcal{A}$. L'ensemble $\mathbf{F}$ des états terminaux de D sont les sous-ensemble de Q qui contiennent au moins un état terminal de $\mathcal{A}$, autrement dit

$\mathbf{F} = \{E \in \mathscr{P}(Q) \mid E \cap F \neq \emptyset\}$. L'automate $\mathcal{D}$ ainsi défini est déterministe complet et il reconnaît exactement le même langage que l'automate $\mathcal{A}$.

La taille de l'automate $\mathcal{D}$ est 2^n. À priori, cet automate n'est pas émondé et possède beaucoup d'états « inutiles ». En pratique, pour n'avoir à traiter que les états « utiles » et ainsi fortement diminuer la taille de l'automate produit, on utilise l'*algorithme de déterminisation* qui produit le *déterminisé* de l'automate. Pour résumer brièvement cet algorithme, on peut dire qu'il produit uniquement la partie accessible de l'automate déterministe décrit ci-dessus.

Voici maintenant une description de l'algorithme. Il commence par construire l'ensemble des états atteints par les transitions sortant des états initiaux I et étiquetées par la lettre a. On appelle P cet ensemble d'état.

L'automate en construction est maintenant composé de deux états : l'ensemble des états initiaux et l'ensemble des états P avec une transition étiquetée par a allant de I vers le nouvel ensemble P. Remarquons que rien n'empêche que $P = I$. On fait de même avec les autres lettres en partant des ensembles déjà construits — dont P et I —. On recommence ainsi le procédé jusqu'à ce que chaque ensemble trouvé possède k transitions sortantes, chacune étiquetée par une des k lettres de A — éventuellement aboutissant vers l'ensemble vide. Le nombre d'ensemble étant majoré par 2^n le processus de construction des états se termine.

Dans l'automate finalement obtenu I est l'état initial, les états terminaux sont les ensembles d'états de Q qui possèdent au moins un état de F.

On peut remarquer que, en plus d'être déterministe et complet, l'automate ainsi construit est accessible.

Généralement on émonde l'automate avant d'appliquer l'algorithme afin de limiter les opérations inutiles lors de son exécution. De plus, en émondant l'automate, on s'assure que l'automate construit est également émondé.

L'algorithme de déterminisation possède une complexité exponentielle dans le pire des cas. On présente, dans la partie 2.1, page 26, l'exemple d'une détermination d'un automate de taille n qui produit un automate déterministe de taille 2^{n-1}.

1.2.7 Minimisation

Soit $\mathcal{A}$ un automate, déterministe ou non, $\mathcal{A} = (Q, A, T, I, F)$, et soit p un état de Q. On appelle le *langage reconnu par l'état* p comme étant le langage reconnu par l'automate $(Q, A, T, \{p\}, F)$. Autrement dit, c'est l'ensemble des mots associés aux chemins partant de l'état $\{p\}$ et aboutissant aux états terminaux F.

Deux états d'un même automate, qui reconnaissent un même langage, sont appelés deux *états équivalents*.

Un automate, déterministe complet, qui ne possède aucune paire d'état équivalents est appelé un *automate minimal*.

On a le résultat remarquable suivant :

Théorème 1 *Si deux automates déterministes complets sont tous les deux minimaux et reconnaissent un même langage alors ils sont isomorphes.*

C'est une propriété remarquable qui permet de définir l'ensemble des automates déterministes minimaux non étiquetés comme un ensemble de représentants canoniques des langages rationnels. Pour un langage $\mathcal{L}L$, on parle volontiers de *l*'automate minimal qui reconnaît $\mathcal{L}L$.

Contrairement à ce que son nom évoque, l'automate minimal n'est pas toujours le plus petit automate qui reconnaît un langage cependant on peut montrer c'est le plus petit, en taille, parmi les automates déterministes.

Soient p, q deux états équivalents d'un automate $\mathcal{A}$. Si $p \neq q$ on peut *fusionner* ces deux états en un unique nouvel état pour produire un nouvel automate qui reconnaît le même langage que $\mathcal{A}$. La fusion de deux états consiste simplement à supprimer l'un de ces deux états, puis à rediriger les transitions arrivant ou sortant de cet état vers l'autre état sans changer le sens de ces transitions.

Il faut remarquer que la fusion de deux états peut transformer un automate déterministe en un automate non déterministe.

Un algorithme de minimisation d'automate est un algorithme qui transforme un automate en l'automate minimal équivalent.

Pour calculer l'automate minimal déterministe complet d'un automate on peut utiliser des algorithmes de minimisation d'automate déterministes. Parmi ceux-ci, les plus connus sont les algorithmes de Moore [Moo56], de Hopcroft [Hop71] et de Brzozowski [Brz62]. L'algorithme de Brzozowski est décrit dans la partie 2.2 page 28. Les algorithmes de Moore et de Hopcroft ne s'appliquent qu'aux automates déterministes alors que l'algorithme de Brzozowski peut servir à produire l'automate déterministe complet minimal d'un automate non-déterministe.

Chapitre 2

Analyse de l'algorithme de Brzozowski

Dans ce chapitre nous étudions la complexité de l'algorithme de Brzozowski. C'est un algorithme de minimisation d'automate qui s'applique aussi aux automates déterministes qu'aux automates non-déterministes. Ici, nous nous intéressons à son utilisation sur les automates déterministes : à priori, il ne semble pas intéressant d'utiliser cet algorithme pour minimiser des automates déterministes car il possède dans le pire des cas une complexité exponentielle alors qu'il existe d'autre algorithmes de complexité polynomiale. Néanmoins il possède la réputation d'être « souvent » efficace en pratique [BBCF10]. Nous allons montrer que ce n'est pas vrai sur les automates déterministes pour la distribution uniforme puisque génériquement il possède une complexité super-polynomiale. Ce travail a été fait en collaboration avec Cyril Nicaud et a été publié dans les actes de la conférence internationale DLT en 2013 [FN13a].

2.1 Motivations

Un algorithme de minimisation d'automate transforme un automate en l'automate minimal déterministe qui reconnaît le même langage. L'automate minimal d'un langage est l'automate qui possède le plus petit nombre d'états parmi les automates déterministes qui reconnaissent le langage. Dans un automate minimal, deux états différents ne peuvent être équivalents car ils seraient redondants.

On préfère manipuler des automates ayant le plus petit nombre d'états possible afin de réduire les coûts de stockage et les complexités des algorithmes qui opèrent sur ces automates. L'ensemble des automates minimaux est une classe de représentants canoniques des langages reconnaissables par automates. Deux automates minimaux reconnaissant un même langage sont isomorphes : ils ne diffèrent que par les étiquettes de leurs états. On peut savoir si deux langages sont égaux en comparant leurs automates minimaux. En combinatoire, on peut utiliser cette propriété pour définir la taille d'un langage rationnel comme étant le nombre d'états de son automate minimal : compter le nombre de langages rationnels d'une certaine taille c'est compter le nombre d'automates minimaux de cette taille.

Pour toutes ces raisons, depuis le début de la théorie des automates, beaucoup de travaux ont été effectués sur les algorithmes de minimisation. Parmi ces algorithmes, les plus connus sont l'algorithme de Moore [Moo56], l'algorithme de

Hopcroft [Hop71] et l'algorithme de Brzozowski [Brz62]. Les deux premiers algorithmes, Moore et Hopcroft, ne s'appliquent qu'aux automates déterministes alors que l'algorithme de Brzozowski est plus général et s'applique aussi aux automates non déterministes. L'algorithme de Brzozowski a de plus l'avantage de pouvoir être facilement implanté en machine car il n'utilise que des opérations classiques et répandues de la théorie des automates comme par exemple l'algorithme qui consiste à émonder un automate et l'algorithme de déterminisation. Sa simplicité et sa généralité en font un algorithme de choix, nous verrons dans la suite qu'il présente quelques lacunes.

L'algorithme de Moore possède une complexité quadratique dans le pire des cas et l'algorithme de Hopcroft, qui est une variante de l'algorithme de Moore, possède, dans le pire des cas, une complexité en $O(n \log n)$. L'algorithme de minimisation de Brzozowski lui, possède dans le pire des cas une complexité exponentielle, et ceci même s'il est appliqué à un automate déterministe. Par exemple l'algorithme de Brzozowski appliqué à un automate déterministe bien choisi mène à la déterminisation de l'automate représenté sur la figure 2.1, et lors de sa déterminisation, le nombre d'états de cet automate passe de n à 2^{n-1}.

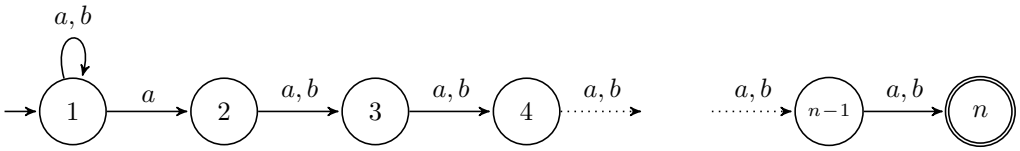


FIGURE 2.1 – Automate dont le miroir est déterministe et dont le déterminisé possède 2^{n-1} états. Dans cet automate, un mot w mène à un sous-ensemble d'état qui dépend des rangs des lettres a dans w. En fait, si on numérote les lettres du mot w en partant de la fin, une lettre a de rang i nous dit que l'état $i+1$ est présent dans le sous-ensemble atteint avec w. Par exemple : les mots a, ba,bba, bb...bba mènent tous au sous-ensemble $\{1, 2\}$; le mot aaa mène à $\{1, 2, 3, 4\}$; le mot ...$abababa$ mène à $\{1, 2, 4, 6, 8, ..., i, i+2, ...\}$. Il existe 2^{n-1} mots de longueur $n-1$ ce qui engendre 2^{n-1} états différents dans le déterminisé.

La grande complexité de cet algorithme dans le pire des cas semble en faire un mauvais candidat pour minimiser un automate déterministe, cependant on peut se poser les questions « Quelle est sa complexité en pratique ? », « Est-ce qu'une telle explosion d'états n'est pas un cas extrêmement rare ? » et « Quelle est sa complexité en moyenne ? ».

On peut illustrer cette question en citant l'algorithme de Quicksort [Hoa62] qui possède une complexité quadratique dans le pire des cas et qui pourtant est très efficace en pratique puisqu'il possède une complexité en moyenne de $O(n \log n)$.

L'article « DFA Minimisation » de l'encyclopédie libre wikipedia le 27 novembre 2013 parle de l'algorithme de Brzozowski — figure 2.1 — et semble suggérer que, expérimentalement, l'algorithme est plus efficace que dans le pire des cas.

On peut consulter l'article complet à l'adresse :

http://en.wikipedia.org/wiki/DFA_minimization

Brzozowski's algorithm [edit]

As Brzozowski (1963) observed, reversing the edges of a DFA produces a non-deterministic finite automaton (NFA) for the reversal of the original language, and converting this NFA to a DFA using the standard powerset construction (constructing only the reachable states of the converted DFA) leads to a minimal DFA for the same reversed language. Repeating this reversal operation a second time produces a minimal DFA for the original language. The worst-case complexity of Brzozowski's algorithm is exponential, as there are regular languages for which the minimal DFA of the reversal is exponentially larger than the minimal DFA of the language,[5] but it frequently performs better than this worst case would suggest.[4]

Il nous a donc semblé intéressant de faire une étude rigoureuse de la complexité en moyenne de l'algorithme de Brzozowski afin de pouvoir le comparer à d'autres algorithmes de minimisation. Les complexités en moyennes des algorithmes de minimisation de Moore et de Hopcroft ont déjà été étudiées.

Un travail de Frédérique Bassino, Julien David et Cyril Nicaud [BDN09, BDN12] montre que l'algorithme de Moore possède une complexité en moyenne en $O(n \log(n))$ lorsque chaque état est terminal ou non terminal selon une distribution de Bernoulli et indépendemment des autres états. Cette étude ne fait aucune hypothèse sur la distribution de la structure de transitions des automates. C'est à dire que, ici, la forme générale de l'automate n'importe pas ; pourvu que le nombre d'états terminaux soit proportionnel à la taille de l'automate, alors le résultat s'applique. En revanche si les automates possèdent peu d'états terminaux ou en nombre constant alors on ne peut rien dire.

Si on fait l'hypothèse supplémentaire que la distribution de la structure de transition est uniforme, Julien David [Dav12, Dav10] a montré que la complexité en moyenne de l'algorithme de Moore est en $O(n \log(\log(n)))$. Julien David a montré également qu'avec cette même distribution, l'algorithme de Hopcroft possède une complexité au moins aussi bonne que celle de Moore lorsque l'ordre de traitement des tâches est sciemment choisi [Dav12].

Nous avons étudié la complexité de l'algorithme de Brzozowski lorsque ses entrées sont des automates déterministes uniformément distribués et plus généralement lorsque la distribution des structures de transitions des automates en entrée est uniforme. La conclusion de cette étude est que pour la distribution uniforme, l'algorithme de Brzozowski n'est pas efficace même en moyenne, et que lorsque cela est possible il vaut mieux utiliser les algorithmes de Moore ou de Hopcroft.

	Pire	Distribution Uniforme
Moore	$O(n^2)$	$O(n \log(\log n)$ en moyenne
Hopcroft	$O(n \log n)$	$O(n \log(\log n))$ en moyenne (pour un certain ordre de traitement).
Brzozowski	$O(2^n)$	génériquement super-polynomial

FIGURE 2.2 – Résumé des complexités des algorithmes de minimisation d'automates déterministes. n est le nombre d'états de l'automate. Les complexités en moyenne sont données lorsque la distribution de la stucture de transition est uniforme.

2.2 Algorithme de Brzozowski

L'algorithme de Brzozowski minimise un automate quelconque, qu'il soit au départ déterministe ou non. Pour cela il utilise le théorème suivant qui est dû à Brzozowski [Brz62].

Théorème 2 Brzozowski *Soit un automate dont le miroir est déterministe et accessible. Alors si lui on applique l'algorithme de déterminisation — voir 1.2.4 — on obtient un automate complet déterministe minimal.*

Preuve du Théorème 2 : Appelons $\mathcal{A}$ l'automate de départ et $\mathcal{B}$ le déterminisé de $\mathcal{A}$. Montrons que deux états équivalents de $\mathcal{B}$ sont égaux. Le miroir $\mathcal{A}$ est accessible et déterministe, appelons 1 son unique état initial et δ sa fonction de transition. L'état 1 est le seul état terminal de $\mathcal{A}$.

Soient P et Q deux états équivalents de $\mathcal{B}$ qui sont donc des ensembles d'états de $\mathcal{A}$. Deux cas peuvent se présenter : ou bien $P = \emptyset$, ou bien $P \neq \emptyset$. Pour commencer supposons que $P \neq \emptyset$.

Soit p un état de P, on va montrer que dans ce cas p est aussi un état de Q. Il existe un chemin allant de p vers 1 l'état terminal de $\mathcal{A}$. Ce chemin est associé à un mot w. Donc w est reconnu par l'état P et ainsi w est reconnu par l'état Q. Il existe donc, dans $\mathcal{A}$, un chemin étiqueté par le même mot w allant d'un état q de Q vers l'état terminal 1.

Donc dans le miroir de $\mathcal{A}$ il existe un chemin allant de 1 vers q et un autre allant de 1 vers p tout deux étiquetés par w' le miroir de w. Or $\mathcal{A}$ est déterministe donc $\delta(1, w') = q = p$.

Ainsi tout état p de P est un état de Q, et comme P et Q jouent des rôles symétriques on montre que $Q = P$.

Supposons maintenant que $P = \emptyset$, alors dans ce cas P et Q ne reconnaissent aucun mot dans $\mathcal{B}$. Par l'absurde supposons que Q ne soit pas vide et qu'il contienne un état q. Comme $\mathcal{A}$ est co-accessible, il existe un chemin allant de q à l'état terminal 1. Donc l'état q reconnaît un mot, donc Q reconnaît un mot, c'est une contradiction. □

De ce lemme on peut déduire l'algorithme de minimisation suivant appelé l'algorithme de Brzozowski. Voici l'algorithme appliqué à un automate $\mathcal{A}$:

1 On émonde l'automate $\mathcal{A}$.

2 On permute les états initiaux et terminaux et on inverse le sens des transitions.

3 On applique l'algorithme de déterminisation — voir 1.2.6 —.

4 On inverse l'automate à nouveau.

5 On lui applique de nouveau l'algorithme de déterminisation.

Théorème 3 *L'automate produit par l'algorithme est déterministe minimal et reconnaît le même langage que l'automate de départ $\mathcal{A}$.*

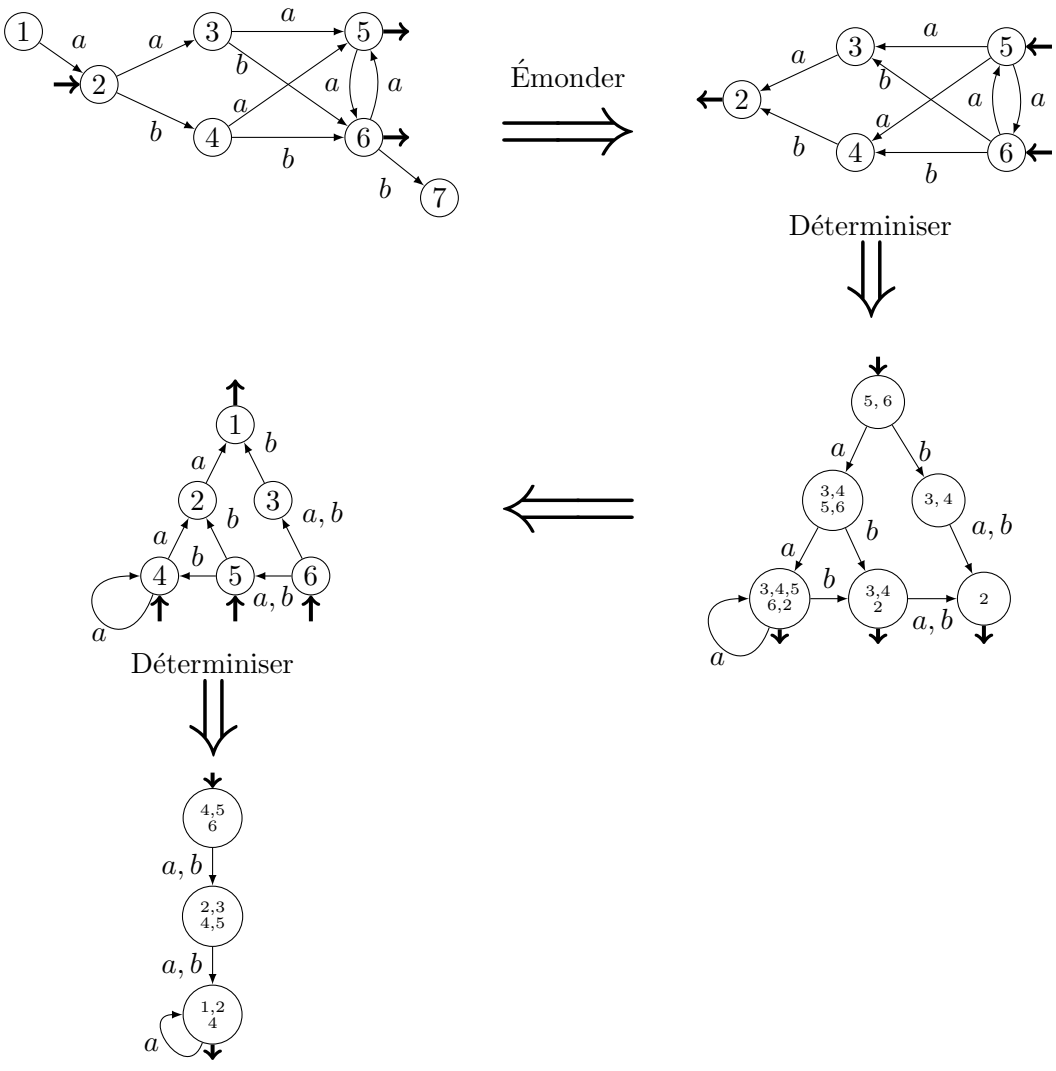


FIGURE 2.3 – Exemple de minimisation d'un automate déterministe avec l'algorithme de Brzozowski. Les automates à droite reconnaissent le langage miroir de l'automate de départ.

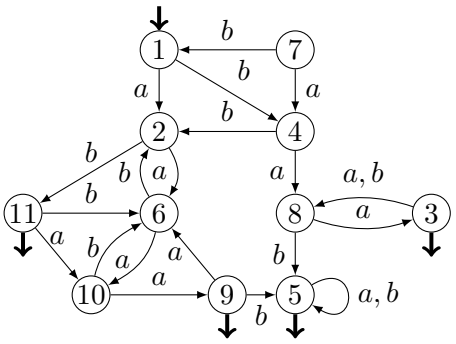
Preuve du Théorème 3 : Lors de l'étape 3 on déterminise l'automate par l'algorithme de déterminisation. Donc l'automate obtenu est déterministe et accessible. L'automate obtenu après l'étape 4 est le miroir du précédent, il vérifie donc les conditions du Théorème 2. Ainsi l'automate obtenu juste après la dernière étape, est déterministe.

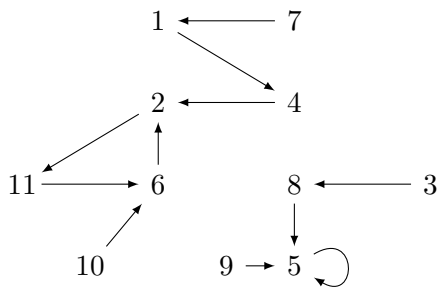
Montrons maintenant que le langage $\mathcal{L}(\mathcal{A})$, reconnu par $\mathcal{A}$, est le même que le langage reconnu par l'automate produit par l'algorithme. Juste après l'étape 1, le langage demeure inchangé. Lors de l'étape 2 l'automate est inversé et reconnaît donc le langage miroir de $\mathcal{L}(\mathcal{A})$. Le langage reconnu reste inchangé après la déterminisation de l'étape 3 puis à l'étape 4 l'automate est inversé de nouveau, le langage reconnu est à nouveau $\mathcal{L}(\mathcal{A})$. Le langage reconnu reste $\mathcal{L}(\mathcal{A})$ après la dernière déterminisation. □

On peut penser que l'étape 1 de l'algorithme, qui consiste à émonder l'automate avant toute déterminisation, n'est pas nécessaire. En effet la preuve précédente est applicable même si l'automate n'est pas émondé et l'automate produit reste bien minimal et reconnaît le même langage. L'algorithme émonde l'automate pour réduire la complexité de la première déterminisation. En fait lorsque l'algorithme émonde l'automate, ce qui importe c'est de retirer les états inaccessibles à partir d'un état source. La présence de ces états peut augmenter le coût en calcul et en mémoire du processus de déterminisation alors que ces états ne contribuent pas au langage reconnu. Émonder l'automate est une opération peu coûteuse : elle a une complexité en $O(n)$. C'est pour cette raison que souvent on émonde un automate avant de le déterminiser.

Lors de la première étape de déterminisation — l'étape 3 — le nombre d'états du déterminisé peut exploser — voir figure 2.1 —. Ensuite le calcul du miroir — étape 4 — de l'automate se fait en un nombre d'opérations de même ordre que celui de la première déterminisation. Le processus qui consiste à émonder — étape 1 — et qui calcul le premier miroir se fait en un nombre d'opérations qui n'excède pas le nombre de transitions c'est à dire kn. La complexité de la seconde déterminisation — étape 5 — dépend du nombre d'états de l'automate produit et de l'automate de départ, or si le nombre d'états de l'automate de départ peut être très grand, l'automate produit possède au plus n états car il est minimal donc plus petit en taille que l'automate du départ. La complexité en temps — et en mémoire — totale de l'algorithme est donc le maximum entre, n, le nombre d'état au départ, et la taille de l'automate aprés l'étape 3 de l'algorithme. Comme nous l'avons fait remarquer précédemment, cette taille peut être exponentiellement grande dans le pire des cas.

La suite de ce chapitre étudie la distribution de la taille de l'automate obtenu après l'étape 3. Nous montrerons que, génériquement, cette taille dépasse tout polynôme en n pour une certaine classe de distributions des entrées de l'algorithme.





$f_b(1) = 4$ $f_b(2) = 11$ $f_b(3) = 8$
$f_b(4) = 2$ $f_b(5) = 5$ $f_b(6) = 2$
$f_b(7) = 1$ $f_b(8) = 5$ $f_b(9) = 5$
$f_b(10) = 6$ $f_b(11) = 6$

FIGURE 2.4 – À gauche un automate déterministe complet avec toutes ses transitions. À droite le graphe de l'application associée à la lettre b.

2.3 Énoncé du théorème

Dans cette section on énonce précisément le résultat que nous avons obtenu puis, on présente les idées principales de sa preuve. La preuve complète est détaillée dans la section suivante.

On s'intéresse à la complexité générique de l'algorithme de Brzozowski sur les automates déterministes distribués uniformément.

Avant de décrire cette distribution nous faisons remarquer que pour tout automate déterministe complet $\mathcal{A}$ de taille n et pour toute lettre c de A on peut associer une application f_c de $[n]$ dans $[n]$. L'application est ainsi définie : pour tout entier naturel i, $i \in [n]$, on pose $f_c(i)$ l'étiquette de l'état qui est relié à l'état d'étiquette i par une transition sortant de ce dernier et étiquetée par la lettre c. Comme l'automate est déterministe une telle transition existe et est unique et donc il n'y a pas d'ambigüité sur l'état vers laquelle cette transition est dirigée. On peut voir ça comme une application de l'ensemble des automates de taille n sur un alphabet de taille k vers l'ensemble des k-uplets de fonctions de $[n]$ dans $[n]$. L'image d'un automate par cette application est appelée la *structure de transition* de l'automate. Cette image ne suffit pas à décrire complètement un automate déterministe complet car elle ne décrit pas l'ensemble des état terminaux.

Sur la figure 2.4 page 31 on peut voir un automate déterministe et l'application associée à la lettre b sous forme de graphe.

La distribution sur laquelle on travaille est la suivante, pour tirer un automate déterministe de taille n, on tire k applications de $[n]$ dans $[n]$. Chacune de ces k applications est tirée uniformément parmi les n^n applications de n dans n, et indépendamment les une des autres. Ces applications vont déterminer les nk transitions de la façon que l'on a décrite précédemment. L'ensemble des états terminaux

est choisi de la façon suivante, chaque état a une probabilité β d'être terminal et ce tirage est indépendant des autre états. On aura toujours $0 < \beta < 1$, et β ne dépend pas de la taille de l'automate n. L'état initial est toujours l'état étiqueté par 1. Le cas où $\beta = \frac{1}{2}$ correspond à la distribution uniforme sur les automates déterministes de taille n.

Définition 1 *Dans la suite nous appellerons cette distribution la distribution* β-uniforme.

Le résultat que nous allons présenter est le suivant :

Théorème 4 *Génériquement, pour la distribution* β*-uniforme, la complexité de l'algorithme de minimisation de Brzozowski dépasse tout polynôme en* n.

Comme nous l'avons fait remarquer en début de section, la complexité de l'algorithme dépend très fortement de l'automate produit après la première déterminisation. Nous allons montrer que cette taille dépasse génériquement tout polynôme en n. La preuve complète de cette proposition est donnée dans la section suivante. La suite de cette section donne seulement les idées principales de cette preuve.

2.3.1 Idée de la preuve

Les états initiaux du miroir sont les états terminaux de l'automate de départ qui n'ont pas été supprimés lors du processus qui consiste à monder l'automate — à l'étape 1 —. C'est à dire que les états initiaux du miroir sont les états terminaux et accessibles de l'automate de départ.

Dans le déterminisé du miroir, l'état initial correspond à l'ensemble des états qui sont, dans l'automate de départ, terminaux et accessibles. Partant de cet ensemble, on va s'intéresser à la suite des ensembles obtenus en lisant systématiquement la lettre a. Les sous-ensembles obtenus ainsi correspondent tous à des états du déterminisé et leur nombre est donc une borne inférieure au nombre d'états. La preuve montre que ce nombre est génériquement suffisamment grand pour dépasser tout polynôme en n avec une probabilité qui tend vers 1 lorsque n grandit.

Prenons maintenant l'ensemble de tous les a-cycles et regardons l'intersection des états de ces cycles avec les images successives de l'ensemble des états initiaux — terminaux dans l'automate du départ —. Ces l'ensemble de ces intersections est appelée la trace des images des états initiaux sur les a-cycles par la lettre a. La taille total de cette trace est une borne inférieure au nombre d'images des états initiaux et donc au nombre d'états du déterminisé. Nous allons donner une borne inférieure à la taille de cette trace.

Chaque a-cycle étant issu du miroir de la structure de transition associée à la lettre a dans l'automate de départ, il n'existe aucune transition étiqueté par la lettre a arrivant dans un a-cycle exceptées celles provenant d'un autre état de ce même cycle. En conséquence, la trace des états initiaux sur un a-cycle est exactement la

suite des images, par la lettre a, des états initiaux présents — initialement — dans ce cycle.

Maintenant intéressons nous à l'image d'un état initial dans un a-cycle et suivons ses images successives par la lettre a. Cet état revient au point de départ lorsqu'on a fait le tour du cycle. C'est à dire au bout d'un nombre d'itérations qui est égal à la longueur du cycle.

Comme chaque état revient à son point de départ au bout d'un nombre d'itérations égal à la longueur du cycle alors la suite des images dans la trace est cyclique avec une période qui divise la longueur du cycle.

Cette période dépend de la configuration des états initiaux dans le cycle. Par exemple si un a-cycle contient uniquement des état initiaux (ou aucun état initiaux), la suite des images est constante et égale à l'ensemble de tous les états du cycle (ou l'ensemble vide). Dans ce cas la période est de 1. Lorsque la période est égale à la longueur du a-cycle alors le cycle est primitif et réciproquement.

À partir de là, on montre facilement que le nombre de valeurs différentes que prend la suite des images des états initiaux par la lettre a est au moins égal au ppcm des périodes de tous les a-cycles accessibles de l'automate. On peut voir sur la figure 2.5 un exemple d'une telle déterminisation. S'il existe suffisamment de a-cycles primitifs accessibles on peut montrer que le ppcm des longueurs de ces cycles dépasse génériquement tout polynôme en n, et donc on montre le théorème 4.

L'idée de la preuve est de regarder les a-cycles de longueur plus grande que $\log n$ et de montrer séparément que :

— génériquement ces cycles sont tous accessibles,
— génériquement ces cycles sont tous primitifs,
— génériquement le ppcm de leur longueurs dépasse tout polynôme en n.

On appelle ces cycles de longueur plus grande que $\log n$ les *grands cycles* de l'automate.

Les grands cycles sont accessibles

Si on tire uniformément deux applications, et que l'on s'intéresse à l'orbite d'un point quelconque par ces deux applications. On s'aperçoit que, avec une probabilité générique, cette orbite possède une taille linéaire en la taille de des deux applications. On peut montrer que cette orbite est de taille supérieure à αn pour une constante α bien choisie qui ne dépend pas de n. La probabilité qu'il existe un cycle de longueur $\log n$ en dehors de l'orbite est environ inférieure à $(1-\alpha)^{\log n}$. Donc un cycle de longueur plus grande que $\log n$ est génériquement accessible.

Les grands cycles sont tous primitifs

On peut voir la succession des états terminaux et non terminaux d'un cycle comme un mot sur deux lettres. Lorsqu'un cycle n'est pas primitif alors ce mot est une puissance d'un mot au moins deux fois plus petit — voir définition de mot primitif —. Cet évènement arrive avec une probabilité exponentiellement faible en la longueur du cycle pour la distribution β-uniforme. En sachant que, génériquement, le nombre de cycles est borné par $K \log n$ où K est une constante, on montre que tous les cycles plus grands que $\log n$ sont primitifs avec une probabilité qui tend

vers 1.

Le ppcm de leur longueurs dépasse tout polynôme en n génériquement

Les états constituant les cycles des transitions a sont appelés les points cycliques de l'application associée à ces transitions. Dans notre modèle de distribution β-uniforme, la distribution de cette application est uniforme. C'est une distribution bien connue ; selon cette distribution le nombre de points cycliques est génériquement supérieur à tout polynôme de la forme n^γ où $0 < \gamma < \frac{1}{2}$.

Si on regarde les points contenus dans les cycles d'une application, on remarque qu'ils forment une permutation ; le ppcm de la longueur de ces cycles est exactement l'ordre — ordre dans le contexte des groupes — de la permutation formée. Pour la loi β-uniforme, la distribution de la permutation formée par les cycles est uniforme lorsque le nombre de points cycliques est fixé.

Un travail d'Erdős et Turán [ET67], sur la distribution uniforme des permutations, permet de montrer que l'ordre d'une permutation de taille n est génériquement supérieur à $n^{\frac{1}{3}\log n}$. Grâce au résultat d'Edmund Landau, sur l'ordre maximum de permutations, on peut montrer que même en ne considérant que les grands cycles, le ppcm est environ $n^{\alpha \log n}$.

Ce résultat et le résultat sur la distribution du nombre de points cycliques nous permet de montrer que le ppcm des longueurs des grands cycles est génériquement super-polynomial.

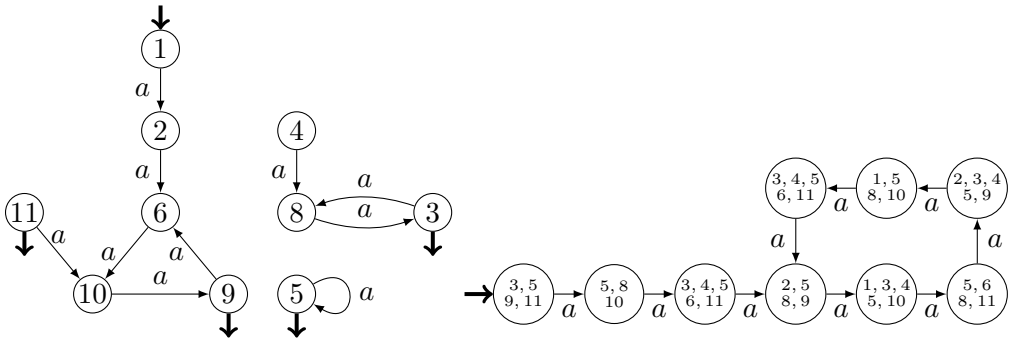


FIGURE 2.5 – À gauche un automate déterministe complet à une seule lettre. À droite la déterminisation de son miroir — sans processus d'émondation préalable —. On remarque que à gauche l'automate possède un a-cycle $(6, 9, 10)$ primitif de longueur 3 et un autre $(3, 8)$ de longueur 2. À droite l'automate possède un cycle de longueur 6 qui est le ppcm de 3 et 2. Le cycle $(8, 3)$ ne peut être rendu accessible que si on ajoute à l'automate une seconde lettre.

Sur la figure 2.6 on présente des observations expérimentales sur les ppcm des longueurs des grands a-cycles accessibles.

2.4 Preuves

Dans la section précédente nous avons présenté les grandes idées de la preuve du théorème 4. Nous allons maintenant donner les détails de cette preuve. Nous

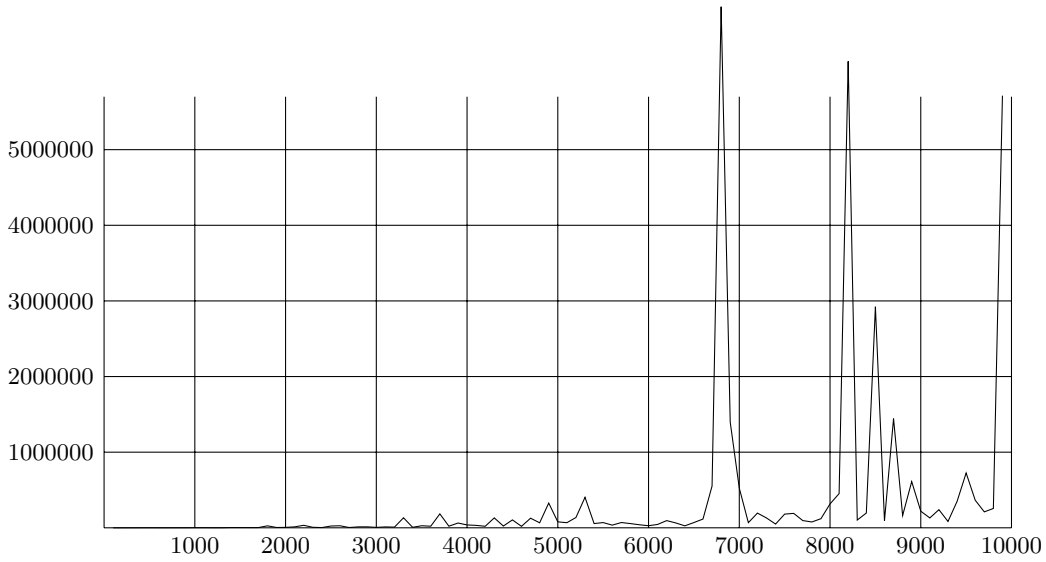


FIGURE 2.6 – Moyenne des ppcm des périodes des a-cycles accessibles et primitifs d'automates à deux lettres tirés uniformément. Les moyennes ont été calculées sur des échantillons de 100 automates. Les tailles des automates générés sont représentées en abscisse, les moyennes des ppcm obtenus sont représentées en ordonnée.

rappelons qu'elle utilise les cycles étiquetés par la lettre a.

On rappelle qu'un cycle est appelé un a-cycle si les transitions qu'il emprunte sont uniquement étiquetées par la lettre a — voir définition a-cycle page **??** — On rappelle aussi que ce cycle est appelé un cycle primitif si le mot à deux lettres définit à partir de la succession des état terminaux et non-terminaux n'est pas une puissance d'un mot strictement plus petit.

Définition 2 *Soit un automate $\mathcal{A}$, c une lettre de $\mathcal{A}$, soit E un sous-ensemble d'états de $\mathcal{A}$ et I l'ensemble des états initiaux de $\mathcal{A}$.*

Pour un ensemble d'état S on pose $\phi(S)$ les états atteignables par des transitions partant de S et étiquetées par la lettre c. On définit alors $\phi^0(I) = I$ et $\phi^n(I) = \phi(\phi^{n-1}(I))$.

On appelle trace des états initiaux par la lettre c sur E, *la suite* $(\phi^n(I) \cap E)_{n\in\mathbb{N}}$

Cette définition ne sert qu'à énoncer le lemme suivant.

Lemme 2 *Soit $\mathcal{A}$ un automate — pas forcément déterministe —, et soit C un a-cycle primitif de $\mathcal{A}$ de longueur ℓ. Si aucun état du cycle ne possède une transition entrante étiqueté par a et provenant d'un état extérieur au cycle, alors la trace des état initiaux par la lettre a sur le cycle C est périodique de période ℓ.*

Lemme 3 *Soit un automate $\mathcal{A}$ déterministe qui contient m a-cycles primitifs et accessibles de longueurs ℓ_1, ℓ_2,..., ℓ_m. On émonde $\mathcal{A}$, on déterminise son miroir. Alors la taille de l'automate obtenu est au moins égale à* $\mathrm{ppcm}(\ell_1, \ell_2, \ldots, \ell_m)$.

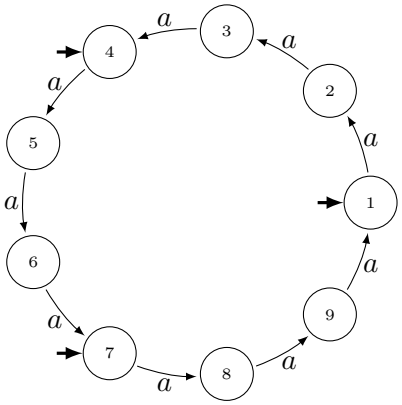


FIGURE 2.7 – Exemple d'un a-cycle de longueur 9 et de période 3. On remarque que la taille de la trace des états initiaux par la lettre a sur les états du cycle est 3 : $\{1,4,7\} \xrightarrow{a} \{2,5,8\} \xrightarrow{a} \{3,6,9\} \xrightarrow{a} \{1,4,7\} \xrightarrow{a} \{2,5,8\} \xrightarrow{a} \dots$.

On prouve ce lemme en observant, lors de la déterminisation, la suite des états obtenue en lisant uniquement la lettre a et en regardant à quel moment on retombe sur un état déjà vu. Regarder la trace de cette suite sur les états des a-cycles suffit pour conclure.

Preuve : Pour $i < m$ on appelle C_i le cycle de longueur ℓ_i. Dans la suite on utilisera indifféremment C_i pour désigner le cycle ou l'ensemble des états par lequel ce cycle passe. On pose $\mathcal{A}'$ que l'on obtient après avoir émondé et renversé l'automate $\mathcal{A}$. On veut montrer que le déterminisé de $\mathcal{A}'$ possède au moins $\text{ppcm}(\ell_1, \ell_2, ..., \ell_m)$ états. Soit I l'ensemble des états initiaux de $\mathcal{A}'$. Comme tous les cycles C_i sont accessibles et primitifs dans $\mathcal{A}$ alors les états de ces cycles ne sont pas supprimés lorsqu'on émonde $\mathcal{A}$ et donc ces cycles sont présents dans $\mathcal{A}'$.

Pour un ensemble E d'états de $\mathcal{A}'$ on pose $\phi(E)$ l'ensemble des états atteints par des transitions étiquetées par a et partant de E. On définit la suite $(\phi^j(I))_{j\in\mathbb{N}}$ par la récurrence suivante : $\phi^0(I) = I$ et $\phi^j(I) = \phi(\phi^{j-1}(I))$ si $j > 0$. Dans le déterminisé de $\mathcal{A}'$ cela correspond à la suite des états parcourus en empruntant uniquement des transitions étiquetées par a et en partant de l'état initial.

On peut faire la remarque suivante : supposons deux entiers s et t qui vérifient $\phi^s(I) \cap C_i = \phi^t(I) \cap C_i$, alors comme les transitions des a-cycles C_i sont issus d'une fonction leurs état ne possède pas de transitions entrante étiquetée par a provenant de l'extérieur du cycle et donc chaque C_i vérifie les hypothèses du lemme 2 page 35. On on déduit que $s - t$ est divisible par la longueur ℓ_i du cycle C_i.

Soient deux entiers s et t tels que $t < s$ et $\phi^s(I) = \phi^t(I)$ alors pour tout cycle C_i on a $\phi^s(I) \cap C_i = \phi^t(I) \cap C_i$ donc pour tout entier i, $s - t$ est divisible par ℓ_i. Donc $s - t$ est divisible par le plus petit multiple commun aux entiers $\ell_1, ..., \ell_m$. Ainsi $\text{ppcm}(\ell_1, ..., \ell_m) \leq s - t$. La proposition contraposée est que pour tout couple d'entiers s, t tel que $t < s$ si $s - t < \text{ppcm}(\ell_1, ..., \ell_m)$ alors $\phi^s(I) \neq \phi^t(I)$. Ce qui

implique que la suite $(\phi^j(I))_{j\in\mathbb{N}}$ prends au moins ppcm$(\ell_1, ..., \ell_m)$ valeurs différentes. Or les $\phi^j(I)$ sont des états du déterminisé, donc le déterminisé de $\mathcal{A}'$ possède au moins ppcm$(\ell_1, ..., \ell_m)$ états différents. □

La suite de la preuve consiste à montrer qu'il y a génériquement suffisamment de a-cycles accessibles et de longueurs suffisamment grandes et variées pour que le ppcm de leur longueurs dépasse n'importe quel polynôme en n. Les deux lemmes qui suivent montrent que si un cycle est suffisamment grand alors la probabilité qu'il ne soit pas accessible est faible.

On commence par énoncer ce premier lemme qui est dû à Arnaud Carayol et Cyril Nicaud [CN12].

Lemme 4 *Il existe un réel α strictement positif qui dépend de la taille de l'alphabet tel que la proportion des états accessibles selon la distribution β-uniforme soit génériquement plus grande que α.*

Comme la partie accessible d'un automate dépend uniquement de sa structure de transitions et de l'état initial, ce lemme s'applique dans tous les cas où la structure de transition est distribuée uniformément et que l'état initial est choisi indépendamment de cette structure. La distribution des états terminaux n'intervient pas ici.

Preuve : Soit un automate déterministe complet $\mathcal{A}$ de taille n. Le sous-automate de $\mathcal{A}$ uniquement composé des états accessibles est un automate déterministe complet accessible.

Sachant cela, pour choisir un automate qui possède une partie accessible de cardinal i il faut, choisir la structure de transitions du sous-automate accessible complet de taille i, choisir les étiquettes des états de ce dernier, et choisir les transitions restantes de façon arbitraire. On rappelle que l'état initial est toujours l'état 1.

On pose T_i le nombre d'automates déterministes complets accessibles de taille i, étiquetés de 1 à i, ayant 1 comme état initial. Soit X_n la variable aléatoire qui compte le nombre d'états dans la partie accessible d'un automate déterministe complet de taille n. On a

$$\mathbb{P}(X_n = i) = \frac{T_i \cdot \binom{n-1}{i-1} \cdot n^{kn-ki}}{n^{kn}}.$$

Dans [Kor78] Korshunov montre que

$$T_i \underset{i}{\sim} E_k \cdot i! \left\{ {ki \atop i} \right\},$$

où E_k dépend de k mais ne varie pas avec i et où $\left\{ {ki \atop i} \right\}$ est un nombre de Stirling de deuxième espèce.

Dans [Goo61] Good donne une approximation asymptotique de $\left\{ {ki \atop i} \right\}$:

$$\left\{ {ki \atop i} \right\} \underset{i}{\sim} \frac{(ki)! \cdot (e^{\rho_k} - 1)^i}{i! \cdot (\rho_k)^{ki} \cdot \sqrt{2\pi ki(1 - ke^{-\rho_k})}},$$

où ρ_k est l'unique solution positive de l'équation en x suivante : $x = k - ke^{-x}$.

En combinant ensemble ces deux formules asymptotiques on obtient

$$T_i \underset{i}{\sim} G\frac{(ki)! \cdot \zeta^i}{\sqrt{i}},$$

où G et ζ ne dépendent pas de i.

On remplace le terme $(ik)!$ par l'équivalent asymptotique, calculé par Stirling, de la suite factorielle, on obtient

$$T_i \underset{i}{\sim} K\gamma^i \cdot i^{ki},$$

où K et γ ne dépendent pas de i.

On remplace le terme T_i dans l'expression de $\mathbb{P}(X_n = i)$. On obtient

$$\begin{aligned}\mathbb{P}(X_n = i) &\underset{i}{\sim} \binom{n-1}{i-1} K\gamma^i \left(\frac{i}{n}\right)^{ki}, \\ &\underset{i}{\sim} \binom{n}{i} \left(\frac{i}{n}\right) K\gamma^i \left(\frac{i}{n}\right)^{ki}.\end{aligned}$$

Soit un réel α, $0 < \alpha < \frac{1}{2}$.

On utilise la majoration suivante : pour tout entier i, $i < \frac{n}{2}$ on a

$$\binom{n}{i} \leq \left(\frac{en}{i}\right)^i.$$

Donc

$$\begin{aligned}\mathbb{P}(X_n \leq \alpha n) &\leq \sum_{i=1}^{\alpha n} \left(\frac{en}{i}\right)^i K\gamma^i \left(\frac{i}{n}\right)^{ki+1} \big(1 + o_i(1)\big), \\ &\leq \sum_{i=1}^{\alpha n} e^i K\gamma^i \left(\frac{i}{n}\right) \left(\frac{i}{n}\right)^{(k-1)i} \big(1 + o_i(1)\big).\end{aligned}$$

Comme dans la somme $i \leq \alpha n$, on a $\frac{i}{n} < \alpha$, donc

$$\begin{aligned}\mathbb{P}(X_n \leq \alpha n) &\leq \sum_{i=1}^{\alpha n} e^i K\gamma^i \left(\frac{i}{n}\right) \alpha^{(k-1)i} \big(1 + o_i(1)\big), \\ &\leq \frac{1}{n} \sum_{i=1}^{\alpha n} i \cdot \big(1 + o_i(1)\big) \Big(e\gamma\alpha^{k-1}\Big)^i K.\end{aligned}$$

Si on choisit α, $0 < \alpha < \frac{1}{2}$ de telle façon que $e\gamma\alpha^{k-1}$ soit inférieur à 1 alors $\mathbb{P}(X_n \leq \alpha n)$ tend vers 0 lorsque n tend vers l'infini. □

Le lemme qui suit est dû à une idée d'Andrea Sportiello.

Lemme 5 *Pour la distribution uniforme des structures de transitions, les a-cycles ayant une longueur supérieure à* $\log(n)$ *sont génériquement tous accessibles.*

Preuve : Pour un entier naturel n, on pose $C_{\log(n)\leq,n}$ l'ensemble des automates de taille n ayant un cycle non-accessible plus grand que $\log(n)$.

Soit X_n la variable aléatoire qui compte le nombre d'états accessibles.

Soit α, $\alpha < 1$ tel que $\mathbb{P}(X_n \leq \alpha n)$ tende vers 0. Un tel α existe voir Lemme 4.

Comme on a

$$\mathbb{P}\big(C_{\log(n)\leq,n}\big) \leq \mathbb{P}\big(X_n \leq \alpha n\big) + \mathbb{P}\big(\alpha n < X_n, C_{\log(n)\leq,n}\big).$$

Il suffit de montrer que $\mathbb{P}(\alpha n < X_n, C_{\log(n)\leq,n})$ tend vers 0.

Soit $C_{\ell,n}$ l'ensemble des automates déterministes de taille n possédant un a-cycle de longueur ℓ dans leur partie qui n'est pas accessible.

On veut majorer $\mathbb{P}(X_n = i, C_{\ell,n})$.

Pour compter le nombre d'automates qui possèdent une partie accessible de cardinal i et qui possèdent un a-cycle non accessible de longueur ℓ il faut choisir les i états accessibles de l'automate, un automate accessible de taille i, les ℓ états du cycle parmi les états non accessibles, les ordonner dans ce cycle et choisir arbitrairement les transitions qui ne sont pas dans le a-cycle.

On remarque que le choix arbitraire des autres transitions peut aboutir à la création d'un autre cycle non accessible de longueur ℓ. Cette façon de compter donne donc une majoration de $\mathbb{P}(X_n = i, C_{\ell,n})$. De plus on suppose que 1 est toujours un état accessible.

Donc

$$\mathbb{P}\big(X_n = i, C_{\ell,n}\big) \leq \frac{\binom{n-1}{i-1} \cdot T_i \binom{n-i}{\ell} \cdot (\ell-1)! \cdot n^{kn-ki-\ell}}{n^{kn}},$$

où T_i est le nombre d'automates déterministes complet accessibles de taille i. De cette formule on déduit la majoration

$$\mathbb{P}\big(X_n = i, C_{\ell,n}\big) \leq \binom{n-i}{\ell} \cdot (\ell-1)! \cdot n^{-\ell} \cdot \mathbb{P}(X_n = i).$$

On a

$$\binom{n-i}{\ell} \cdot (\ell-1)! \cdot n^{-\ell} = \frac{(n-i)!}{(n-i-\ell)! \cdot \ell} \cdot n^{-\ell} \leq (n-i)^{\ell} \cdot n^{-\ell}.$$

Donc

$$\mathbb{P}(X_n = i, C_{\ell,n}) \leq (n-i)^{\ell} \cdot n^{-\ell} \cdot \mathbb{P}(X_n = i).$$

$$\begin{aligned}\mathbb{P}(\alpha n < X_n, C_{\log(n)\leq,n}) &= \sum_{\substack{\log(n)\leq\ell \\ \alpha n\leq i}} \mathbb{P}(X_n = i, C_{\ell,n}),\\ &\leq \sum_{\log(n)\leq\ell} (n-\alpha n)^{\ell} n^{-\ell} \sum_{\alpha n\leq i} \mathbb{P}(X_n = i),\\ &\leq \mathbb{P}(\alpha n \leq X_n) \sum_{\log(n)\leq\ell} (1-\alpha)^{\ell},\\ &\leq \sum_{\log(n)\leq\ell} (1-\alpha)^{\ell}.\end{aligned}$$

Cette dernière majoration est le reste d'une série convergente, on a donc que la quantité $\mathbb{P}(\alpha n < X_n, C_{\log(n)\leq,n})$ tend vers 0 lorsque n tend vers l'infini. □

L'accessibilité des cycles ne dépend que de la structure de transition d'un automate, c'est pourquoi la preuve ne traite pas la distribution des états terminaux.

Lemme 6 *Pour tout réel α, $0 \leq \alpha < \frac{1}{2}$, le nombre de points cycliques d'une fonction uniforme de n dans n est génériquement plus grand que n^{α}.*

Les fonctions aléatoires de $[n]$ dans $[n]$ sont des structures classiques de combinatoire analytique et de probabilités discrètes. Ce résultat peut-être obtenu par des méthodes de combinatoire analytique. Pour plus d'information sur ce genre de méthode on peut consulter [FS09] ou [FO89]. La preuve utilisée ici contient uniquement des calculs de probabilités discrètes. On regarde la suite des images d'un entier 1 et on montre que la partie cyclique de son orbite est génériquement plus grande que n^{α}.

Preuve : Soit f une application de $[n]$ dans $[n]$ tirée de façon uniforme. On définit la suite $\left(f^{\ell}(1)\right)_{\ell\in\mathbb{N}}$ par la récurrence suivante $f^0(1) = 1$ et $f^{\ell}(1) = f(f^{\ell-1}(1))$ pour $\ell > 0$. La suite prend ses valeurs dans un ensemble fini. Il existe donc deux entiers naturels i et j, $i < j$ tels que $f^i(1) = f^j(1)$. On prend un tel j et un tel i les plus petits possibles. Les points cycliques de la suite sont les points $f^{\ell}(1)$ tels que $i \leq \ell < j$. Soient Y_n la variable aléatoire de la valeur de j et X_n la variable aléatoire de la valeur de i. On veut montrer que la suite $\mathbb{P}(Y_n - X_n \leq n^{\alpha})$ tend vers 0 lorsque n tend vers l'infini. Soit un entier naturel m, $m \leq n$. Pour que la valeur Y_n soit supérieure ou égale à m il faut et il suffit que pour tout entier naturel t, $t < m$ on ait $f^t(1) \notin \{f^0(1), f^1(1), \ldots, f^{t-1}(1)\}$. Donc

$$\mathbb{P}(m \leq Y_n) = \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\ldots\left(1 - \frac{m-1}{n}\right).$$

On a évidemment $\mathbb{P}(X_n = k \mid Y_n = m) = \frac{1}{m+1}$ pour $0 \leq k \leq m$.

Montrons que pour tout réel β qui vérifie $\alpha < \beta < \frac{1}{2}$ on a génériquement $n^{\beta} \leq Y_n$.

La fonction $\exp(-2x)$ est convexe et son image en $x = \frac{1}{2}$ — resp en 0 — est inférieure à l'image de $1-x$ en $\frac{1}{2}$ — resp. en 0 —. Donc pour tout réel x, $0 \leq x \leq \frac{1}{2}$ on a $\exp(-2x) \leq 1 - x$.

Si $m < \frac{n}{2}$, en utilisant cette inégalité on a

$$\begin{aligned}
\mathbb{P}(m \leq Y_n) &\geq \exp\left(-\frac{2 \cdot 1}{n}\right) \cdot \exp\left(-\frac{2 \cdot 2}{n}\right) \ldots \exp\left(-\frac{2 \cdot (m-1)}{n}\right), \\
&\geq \exp\left(-\frac{2}{n} \sum_{t=1}^{m-1} t\right), \\
&\geq \exp\left(-\frac{m(m-1)}{n}\right).
\end{aligned}$$

Puisque $n^\alpha < \frac{n}{2}$ à partir d'un certain rang de n, on suppose que l'on dépasse ce rang et on remplace m par n^α dans la formule.

On obtient alors $\frac{n^{2\alpha} - n^\alpha}{n}$ qui tend vers 0 car $\alpha < \frac{1}{2}$.

Donc $\mathbb{P}(n^\alpha \leq Y_n)$ tend vers 1 et $\mathbb{P}(Y_n < n^\alpha)$ tend vers 0.

Soit un entier naturel k, $k \leq n^\beta$ alors

$$\begin{aligned}
\mathbb{P}(X_n = k \mid n^\beta \leq Y_n) &= \sum_{t=n^\beta}^{n} \frac{\mathbb{P}(X_n = k \mid Y_n = t)\mathbb{P}(Y_n = t)}{\mathbb{P}(n^\beta \leq Y_n)}, \\
\mathbb{P}(X_n = k \mid n^\beta \leq Y_n) &\leq \frac{1}{n^\beta} \sum_{t=n^\beta}^{n} \frac{\mathbb{P}(Y_n = t)}{\mathbb{P}(n^\beta \leq Y_n)}.
\end{aligned}$$

Ainsi $\mathbb{P}(X_n = k \mid n^\beta \leq Y_n) \leq \frac{1}{n^\beta}$ pour tout $k \leq n^\beta$. Donc

$$\mathbb{P}(Y_n - X_n \leq n^\alpha \mid n^\beta \leq Y_n) \leq \frac{n^\alpha}{n^\beta}.$$

Or comme $\alpha < \beta$ alors $\mathbb{P}(Y_n - X_n \leq n^\alpha \mid n^\beta \leq Y_n)$ tend vers 0.

Comme nous avons

$$\mathbb{P}(Y_n - X_n \leq n^\alpha) \leq \mathbb{P}(Y_n - X_n \leq n^\alpha \mid n^\beta \leq Y_n) \cdot \mathbb{P}(n^\beta \leq Y_n) + \mathbb{P}(Y_n < n^\beta),$$

alors $\mathbb{P}(Y_n - X_n \leq n^\alpha)$ tend vers 0. □

Lemme 7 *Soit n un entier naturel non nul et soient σ et τ deux permutations de même taille. On tire une fonction de n dans n uniformément. Alors :*

- *Les points cycliques de la fonction de n dans n après avoir été normalisés forment une permutation.*
- *La probabilité que cette permutation soit égale à σ est la même que celle d'être égale à τ.*

Preuve : Une fonction de n dans n peut être vue comme une permutation d'arbres de Cayley telle que la somme des tailles des arbres soit égale à n. Cette permutation est aussi la permutation des points cycliques de la fonction.

On peut définir une application qui à une fonction de n dans n associe un ensemble d'arbres de Cayley et une permutation ayant pour taille le cardinal de cet ensemble — voir figure 2.8 page 42 —.

Cette application est bijective, c'est à dire que n'importe quel couple formé d'une permutation de taille i et d'un ensemble d'arbres de Cayley de cardinal i est l'image d'une une unique fonction f par cette application. La taille de f est la taille de l'ensemble d'arbre au sens combinatoire c'est à dire la somme des noeuds de tous les arbres de l'ensemble.

Donc à permutation fixée de taille i, si l'on souhaite compter le nombre de fonctions de $[n]$ dans $[n]$ dont les points cycliques forment cette permutation, il suffit de compter le nombre d'ensembles d'arbres de Cayley de cardinal i et de taille n.

Ainsi, pour la distribution uniforme sur les fonctions de $[n]$ dans $[n]$, deux permutations de même taille ont une même probabilité d'être tirée. □

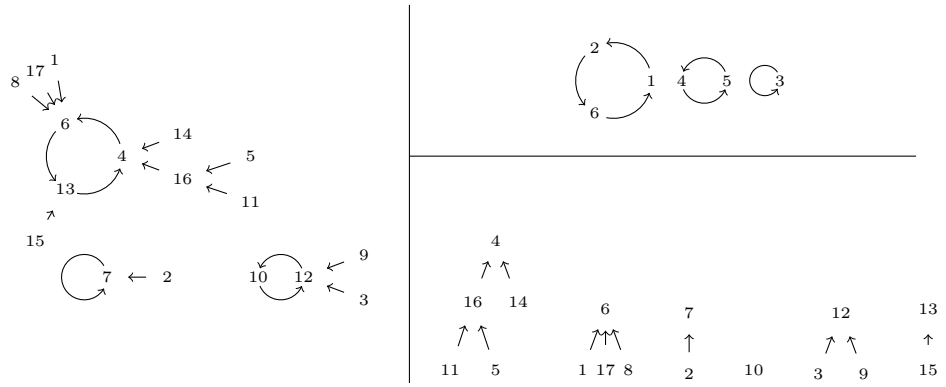


FIGURE 2.8 – À gauche le graphe d'une fonction. À droite sa décomposition en un ensemble ordonné d'arbres de Cayley et en une permutation.

Lemme 8 *Génériquement, une permutation de taille N possède moins de $2\log(N)$ cycles pour la distribution uniforme sur les permutations de taille N.*

Preuve : Soit X_N la variable aléatoire du nombre de cycles d'une permutation uniforme. L'espérance $E(X_N)$ du nombre de cycles d'une permutation est équivalent à $\log(N)$ — voir [FS09, p. 644] —. L'écart type $\sigma(X_N)$ du nombre de cycles est égal à $\sqrt{\log(N)}$.

$$\begin{aligned}\mathbb{P}(X_N \geq 2\log(N)) =& \mathbb{P}(X_N - \log(N) \geq \log(N)),\\ \leq& \mathbb{P}(|X_N - E(X_N)| \geq \log(N)).\end{aligned}$$

En utilisant l'inégalité de Bienaymé-Tchebychev on peut écrire

$$\mathbb{P}(|X_N - E(X_N)| \geq \log(N)) \leq \frac{\sigma^2(X_N)}{\log^2(N)}.$$

Donc $\mathbb{P}(X_N < 2\log(N))$ tend vers 1. □

Lemme 9 *Pour la distribution β-uniforme, $0 < \beta < 1$, tous les cycles de longueur plus grande que* $\log(n)$ *sont primitifs avec une probabilité qui tend vers* 1.

Preuve : Soit un cycle de longueur ℓ. On part de l'état du cycle qui possède l'étiquette la plus petite et on lit, dans l'ordre des a-transitions, tous les états du cycle. Chacun de ces états peut être terminal ou non terminal. En associant les états terminaux à une lettre x et les états non terminaux à une autre lettre y, la succession de ces états est associée à un mot w. Ce mot, de longueur ℓ, ne contient donc que des lettres x ou y.

Posons X_ℓ la variable aléatoire associée au mot w lu sur un cycle.

La probabilité $\mathbb{P}(X_\ell = w)$ dépend uniquement de son nombre d'état terminaux et non terminaux. Si son nombre d'état terminaux est k alors cette probabilité est égale à

$$\mathbb{P}(X_\ell = w) = \beta^k (1-\beta)^{\ell-k}.$$

Le cycle n'est pas primitif si le mot peut s'écrire comme une puissance d'un mot plus petit u. C'est à dire s'il existe un entier naturel d, $d \neq 1$ tel que $u^d = w$. S'il existe un tel d alors évidemment d divise ℓ.

Soit E_ℓ l'évènement : le cycle de longueur ℓ n'est pas primitif.

On a

$$\mathbb{P}(E_\ell) \leq \sum_{\substack{d|\ell \\ d\neq 1}} \sum_{u \in A^{\frac{\ell}{d}}} \mathbb{P}(X_\ell = u^d).$$

En partitionnant l'ensemble des mots de taille $\frac{\ell}{d}$ selon i : leur nombre de lettre x on obtient

$$\mathbb{P}(E_\ell) \leq \sum_{\substack{d|\ell \\ d\neq 1}} \sum_{0 \leq i \leq \frac{\ell}{d}} \binom{\frac{\ell}{d}}{i} \beta^{id} (1-\beta)^{(\frac{\ell}{d}-i)d},$$

$$\mathbb{P}(E_\ell) \leq \sum_{\substack{d|\ell \\ d\neq 1}} \left(\left(\beta^d + (1-\beta)^d \right)^{\frac{1}{d}} \right)^\ell.$$

On pose $||\beta||_q = (\beta^q + (1-\beta)^q)^{\frac{1}{q}}$.

On utilise l'inégalité suivante : si $1 < p \leq q$ alors $||\beta||_q \leq ||\beta||_p$.

Comme pour tout diviseur d on a $2 \leq d \leq \ell$, en remplaçant dans la formule on a

$$\mathbb{P}(E_\ell) \leq \sum_{\substack{d|\ell \\ 1\neq d}} ||\beta||_d^\ell \leq \ell ||\beta||_2^\ell.$$

Or pour tout réel β, $0 < \beta < 1$ on a $||\beta||_2 < 1$. On pose $\lambda = ||\beta||_2$. La probabilité qu'un cycle de longueur plus grande que $\log(n)$ soit primitif est donc supérieure à $1 - \log(n)\lambda^{\log(n)}$.

En supposant qu'il n'y ait pas plus de $2\log(n)$ a-cycles, le tirage des états terminaux étant indépendant du tirage de la structure de transition, la probabilité que tous les cycles plus grand que $\log(n)$ soient primitifs est supérieure à

$$\left(1 - \log(n)\lambda^{\log(n)}\right)^{2\log(n)}.$$

Cette fonction tend vers 1 lorsque n tend vers l'infini.

D'après le lemme 8 il y a génériquement pas plus de $2\log(n)$ a-cycles. Donc la probabilité qu'il n'y ait pas plus de $2\log(n)$ a-cycles et que les cycles de longueur supérieure à $\log(n)$ soient primitif tend vers 1 lorsque n tend vers l'infini. À fortiori la probabilité que tous les a-cycles de longueur supérieure à $\log(n)$ soient primitifs tend vers 1. □

Le lemme suivant est dû à Pál Erdős et à Pál Turán.

Lemme 10 Erdős et Turán. *Pour la distribution uniforme, l'ordre d'une permutation de taille n est génériquement plus grande que $n^{\frac{1}{3}\log n}$.*

Preuve : L'article d'Erdős et de Turán [ET67] donne des résultats bien plus précis. Ici utilise que la distribution du logarithme de l'ordre d'une permutation de taille n a pour moyenne $\frac{1}{2}\log^2 n$ et pour variance $\frac{1}{3}\log^3 n$ et que cette distribution centrée réduite converge en loi vers la loi normale. □

On appelle un *grand cycle* d'une permutation uniforme, un cycle dont la longueur dépasse $3\log(\ell)$, où ℓ est la taille de la permutation.

Lemme 11 *Dans une permutation aléatoire uniforme de taille ℓ le ppcm des longueurs des* **grands cycles** *de cette permutation dépasse génériquement* $\exp(\frac{1}{4}\log^2(\ell))$.

Preuve : Soit une permutation σ de taille ℓ. On sépare les cycles de cette permutation en deux ensembles. On pose P l'ensemble des cycles de longueur plus petite que $3\log(\ell)$ et on pose G l'ensemble des cycles restants. Pour un ensemble de cycle E on appelle $L(E)$ le ppcm des longueur de ses cycles. Le ppcm des cycles σ est égal à $L(P \cup G)$. Or on a $L(P \cup G) \leq L(P) \cdot L(G)$ donc $\frac{L(P \cup G)}{L(P)} \leq L(G)$.

On note γ_ℓ la borne supérieure du ppcm des longueurs des cycles des permutations de taille ℓ. Un résultat de Landau montre que le logarithme de γ_ℓ est asymptotiquement équivalent à la suite $\sqrt{\ell \log \ell}$. On en déduit que le ce ppcm est borné supérieurement par la fonction $\exp\left(2\sqrt{\ell \log \ell}\right)$ à partir d'un certain rang ℓ.

L'ensemble des cycles de P forment une permutation. D'après le lemme 8 le nombre de cycles d'une permutation de taille ℓ est génériquement inférieur à $2\log \ell$ donc la permutation formée par les cycles de P est de taille inférieure à $6\log^2 \ell$. En appliquant le résultat de Landau on en déduit que génériquement le ppcm des longueurs de ces cycles est borné supérieurement par $\exp\left(2\sqrt{6\log^2 \ell \log(\log^2 \ell)}\right)$.

Le lemme 11 nous dit que génériquement $\exp(\frac{1}{3}\log^2 \ell) \leq L(P \cup G)$.
Donc on a $L(G) \geq \frac{L(P \cup G)}{L(P)} \geq \frac{\exp(\frac{1}{3}\log^2 \ell)}{\exp\left(2\sqrt{6\log^2 \ell \ \log(6\log^2 \ell)}\right)} \ L(G) \geq \exp(\frac{1}{4}\log^2 \ell)$ □

Lemme 12 *Dans une fonction aléatoire uniforme de taille n, le ppcm des cycles de longueur plus grande que* $\log(n)$ *est génériquement plus grand que tout polynôme en n.*

Preuve : Soit F_n la variable aléatoire qui prend ses valeurs dans $\mathfrak{F}_n$ — l'ensemble des fonctions de taille n — et qui suit la distribution uniforme sur $\mathfrak{F}_n$. C'est à dire que pour n'importe quelle fonction $f \in \mathfrak{F}_n$, $\mathbb{P}(f = F_n) = \frac{1}{|\mathfrak{F}_n|}$. Soit G_n la variable aléatoire qui prend ses valeurs dans $\mathfrak{G}_n$ — l'ensemble des permutations de taille n — et qui est uniforme sur $\mathfrak{G}_n$. C'est à dire que pour n'importe quel permutation σ, $\sigma \in \mathfrak{G}_n$, $\mathbb{P}(\sigma = G_n) = \frac{1}{|\mathfrak{G}_n|}$. Pour une permutation σ on pose $L_r(\sigma)$ le ppcm des longueurs des cycles plus longs que r et $|\sigma|$ la taille de la permutation. Soit f la fonction qui à un entier naturel n associe le réel $\exp(\frac{1}{4}\log^2(n))$. Pour une fonction g de $[n]$ dans $[n]$ on associe $\mathbf{Cy}(g)$ la permutation normalisée associée aux points cycliques de g.

On veut montrer que, pour tout entier naturel p, $\mathbb{P}(L_{\log(n)}(\mathbf{Cy}(F_n)) > n^p)$ tend vers 1 lorsque n tend vers l'infini.

D'après le lemme 7 pour toute permutation σ de taille ℓ, $\ell \leq n$ on a

$$\mathbb{P}\big(\mathbf{Cy}(F_n) = \sigma \mid |\mathbf{Cy}(F_n)| = \ell\big) = \mathbb{P}(G_\ell = \sigma).$$

Donc on en déduit que pour tout couple d'entiers naturels non nuls n et ℓ on a

$$\mathbb{P}\big(f(\ell) \leq L_{3\log(\ell)}\big(\mathbf{Cy}(F_n)\big) \mid |\mathbf{Cy}(F_n)| = \ell\big) = \mathbb{P}\big(f(\ell) \leq L_{3\log(\ell)}(G_\ell)\big).$$

On définit la suite $(m_\ell)_{\ell \in \mathbb{N}}$ de la manière suivante :

$$m_\ell = \operatorname*{Inf}_{\ell \leq i} \mathbb{P}\left(f(i) \leq L_{3\log(i)}(G_i)\right).$$

D'après le lemme 11 nous savons que $\mathbb{P}(f(i) \leq L_{3\log(i)}(G_i))$ tend vers 1 lorsque ℓ tend vers l'infini, donc m_ℓ tend vers 1 également. De plus m_ℓ est croissante.

On suppose que $n^{\frac{1}{3}} \leq \ell$; on a que $\log(n) \leq 3\log(\ell)$ et donc que

$$m_{n^{\frac{1}{3}}} \leq \mathbb{P}\left(f(n^{\frac{1}{3}}) < L_{\log(n)}(G_\ell)\right).$$

Calculons la limite de $\mathbb{P}\left(L_{\log(n)}(\mathbf{Cy}(F_n)) > f(n^{\frac{1}{3}})\right)$ lorsque n tend vers l'infini.

$$\begin{aligned}
\mathbb{P}\left(L_{\log(n)}(\mathbf{Cy}(F_n)) > f(n^{\frac{1}{3}})\right) &\geq \sum_{n^{\frac{1}{3}}<\ell} \mathbb{P}\left(L_{\log(n)}(\mathbf{Cy}(F_n)) > f(n^{\frac{1}{3}}), |\mathbf{Cy}(F_n)| = \ell\right) \\
&\geq \sum_{n^{\frac{1}{3}}<\ell} \mathbb{P}\left(L_{\log(n)}(\mathbf{Cy}(F_n)) > f(n^{\frac{1}{3}}) \mid |\mathbf{Cy}(F_n)| = \ell\right) \cdot \mathbb{P}(|\mathbf{Cy}(F_n)| = \ell) \\
&\geq \sum_{n^{\frac{1}{3}}<\ell} m_\ell \mathbb{P}(|\mathbf{Cy}(F_n)| = \ell) \\
&\geq m_{n^{\frac{1}{3}}} \sum_{\ell > n^{\frac{1}{3}}} \mathbb{P}(|\mathbf{Cy}(F_n)| = \ell) \\
&\geq m_{n^{\frac{1}{3}}} \mathbb{P}(n^{\frac{1}{3}} \leq |\mathbf{Cy}(F_n)|).
\end{aligned}$$

Or on sait que $m_{n^{\frac{1}{3}}}$ tend vers 1 et comme $\frac{1}{3} \leq \frac{1}{2}$ le nombre de points cycliques est génériquement plus grand que $n^{\frac{1}{3}}$ — voir lemme 6 — donc $\mathbb{P}\left(n^{\frac{1}{3}} \leq |\mathbf{Cy}(F_n)|\right)$ tend vers 1 également. De plus $f\left(n^{\frac{1}{3}}\right) = \exp\left(\frac{1}{36}\log^2(n)\right) = n^{\frac{1}{36}\log(n)}$ et dépasse donc tout polynôme en n. Donc génériquement le ppcm des cycles de longueur plus grande que $\log(n)$ dépasse tout polynôme en n. □

Preuve du Théorème 4 : On s'interesse aux a-cycles de longueur supérieure à $\log(n)$ selon la distribution β-uniforme. Le lemme 5 montre que génériquement ces cycles sont tous accessibles. Le lemme 9 montre que génériquement ces cycles sont tous primitifs. Le lemme 12 montre que génériquement le ppcm des longueurs de ces cycles dépasse tout polynôme lorsque n tend vers l'infini. Alors selon le lemme 3, le determinisé du miroir d'un automate de taille n tiré selon la distribution β-uniforme possède un nombre d'état qui dépasse génériquement tout polynôme en n. Donc la complexité cette déterminisation, qui est l'étape 4 de l'algorithme, dépasse tout polynôme en n. □

2.5 Perspectives sur l'algorithme de Brzozowski

La distribution étudié engendre un grand nombre d'état terminaux, génériquement en quantité linéaire. On peut se poser la question que se passe t-il si la probabilité pour un état d'être terminal varie avec la taille de l'automate ? Par exemple $\frac{1}{\log n}$ ou bien si le nombre d'état terminaux est constant.

Cette question, qui se pose aussi pour les autres algorithmes de minimisation — [Dav12], [Dav10], [BDN09], [BDN12] — est pour l'instant ouverte. En ce qui concerne l'étude de l'algorithme de Brzozowski présenté dans ce chapitre on peut adapter une partie des preuves à condition d'ajouter une lettre à l'alphabet et montrer qu'en moyenne la complexité de l'algorithme reste super-polynomiale même s'il n'y a qu'un seul état terminal. C'est un travail en cours de finalisation ce qui explique qu'il n'apparaît pas dans ce manuscrit. Les principales étapes de la preuve pour un alphabet de cardinal 3 sont :

— Il existe souvent un arbre pour la lettre b de taille au moins $2n/3$.
— Cet arbre est souvent co-accessible car génériquement dans l'automate il y a une unique composante fortement connexe stable, qui en plus contient plus de la moitié des états.
— On en déduit qu'il existe souvent un mot u tel que si on lit u depuis l'état initial de l'automate miroir l'ensemble d'état obtenus contient la racine du grand b-arbre.
— En utilisant les propriétés typiques des arbres et des fonctions aléatoires on en déduit qu'il existe un entier i tel que si on lit ub^i depuis l'état initial du miroir on atteint un ensemble possédant $\sqrt{n}$ éléments.
— Cet ensemble joue le même rôle que des états terminaux rendant les a-cycles primitifs : on peut trouver d a-cycles de longueur à peu prés $\sqrt{n}$ qui sont primitifs.
— Pour conclure la preuve on montre que le ppcm de ces a-cycles est suffisamment souvent supérieur à $C \cdot \sqrt{n}^d$.

Une autre question naturelle est de ne considérer que les automates accessibles. D'un point de vu combinatoire ce sont des études plus compliquées ; il y a néanmoins une technique décrite dans l'article [CN12] qui permet d'établir des résultats sur les automates accessibles à partir de résultats sur les automates généraux. Il semblerait que cela fonctionne encore une fois dès que l'on a un alphabet de cardinal supérieur ou égal à 3. C'est également un travail en cours.

Si ces travaux aboutissent on pourra directement en déduire que la complexité en moyenne en état de l'opération miroir est super-polynomiale. Cela signifie que si on choisi un langage rationnel dont l'automate minimal est de taille n uniformément au hasard, le nombre d'états de l'automate minimal de son miroir est super-polynomial en n.

Dans un autre ordre d'idée, puisque l'algorithme de Brzozowski fonctionne sur des entrées non déterministes, il serait pertinent d'étudier son comportement sur des automates non déterministes aléatoires. Cela présente un problème majeur car il n'y a pas de « bon » modèle connue. Une étude expérimentale a été faite par Tabakov et Vardi [TV05] : ils ont étudiés une distribution de type Erdős-Rényi [ER59] sur les automates non déterministes, en comparant l'efficacité de l'algorithme de Brzozowski à l'algorithme consistant à d'abord déterminiser puis à appliquer l'algorithme de minimisation d'Hopcroft.

D'après leurs résultats expérimentaux il semblerait que les deux algorithmes sont compétitifs, et qu'il vaut mieux utiliser l'algorithme de Brzozowski dès qu'il y a plus de $1,5 \cdot n$ transitions et l'autre algorithme sinon.

Il serait très intéressant mais probablement difficile de donner une explication rigoureuse de ces observations.

Automates acycliques

Dans les deux chapitres suivants nous proposons deux méthodes de génération aléatoire d'automates acycliques. Un automate acyclique est un automate dont le graphe sous-jacent ne possède aucun cycle. L'ensemble des automates acycliques est exactement l'ensemble des automates émondés qui reconnaissent un nombre fini de mots.

Ils sont donc notamment utilisés en traitement automatique des langues : ils permettent de représenter de façon compacte un dictionnaire tout en permettant un test d'appartenance linéaire en la longueur des mots cherchés.

Notamment il peut être pertinent, dans ce contexte, de définir la taille d'un ensemble de mot comme la taille de l'automate minimal acyclique qui reconnaît cet ensemble.

Ces applications naturelles ont suscitées de nombreux travaux sur cette catégorie d'automates [CCM09],[CCM10]. Le résultat le plus célèbre est sûrement l'algorithme de minimisation de Dominique Revuz qui miniminise un automate acyclique en un nombre linéaire d'opérations [Rev92].

D'un point de vue plus combinatoire, plusieurs travaux se sont intéressés à l'énumération et à la génération de ces automates.

Liskovets donne une formule d'inclusion-exclusion [Sta00] qui compte le nombre d'automates acycliques [Lis06], malheureusement la somme alternée rend l'estimation asymptotique difficile. Voici la formule ; $\alpha(n)$ compte le nombre d'automates acycliques de taille n.

$$\alpha(n) = \sum_{t=0}^{n-1} \binom{n}{t} (-1)^{n-t-r} (t+1)^{k(n-t)} \alpha(t). \tag{2.1}$$

Une preuve de ce type de formule est donnée page 89.

Michael Domaratzki a donc tenté une autre approche pour encadrer ce nombre — [Dom02],[DKS01] —.

Cependant, les formules qu'il propose également mal adaptées pour faire de l'asymptotique, et ne sont même pas très précises puisque les bornes supérieure et inférieure différent d'un facteur exponentiel.

Concernant la génération, le seul résultat disponible à notre connaissance est dû à Almeida, Moreira et Reis qui ont publiés une manière de générer de manière exhaustive tous les automates acycliques d'une taille donnée — [AMR08] —.

Malheureusement le nombre d'automates croissant très vite avec la taille n, on peut difficilement dépasser $n = 11$ ou $n = 12$.

Nous avons donc cherché à construire des générateurs aléatoires d'automates acycliques et d'automates acycliques minimaux. Il peuvent être utilisés pour tester

n	$s = 1, 2, 3, \ldots$	$s = 1$	$s = 2$	$s = 3$	$s = 4$
0	1	0	0	0	0
1	1	1	0	0	0
2	7	6	1	0	0
3	142	96	45	1	0
4	5 941	3 048	2 640	252	1
5	428 856	163 560	218 100	45 920	1 275
6	47 885 899	13 412 160	24 870 960	8 938 440	658 200
7	7 685 040 448	1 568 700 000	3 792 703 320	2 032 357 320	282 742 950

FIGURE 2.9 – Le tableau de gauche énumère le nombre total d'automates acycliques en fonction du nombre d'états. Les tableaux de droite donnent le nombre d'automates acycliques en fonction du nombre d'états et du nombre de sources. Le cardinal de l'alphabet est 2.

empiriquement des performances d'algorithmes sur ces automates, mais également pour observer la forme typique d'un automate acyclique de grande taille et assister de futures analyses théoriques.

Propriétés sur les automates acycliques

Un automate acyclique est défini comme un automate déterministe — non complet — dont le graphe sous-jacent est acyclique.

Lemme 13 *Soit $\mathcal{A}$ un automate acyclique de taille n. Soit un chemin ..., e_{-1}, t_{-1}, e_0, t_1, e_1, ... de $\mathcal{A}$. Alors la longueur de ce chemin est finie et passe au plus par n états et tous ces états sont deux à deux différents.*

Cela implique que tout chemin d'un automate acyclique emprunte au plus $n-1$ transitions et ces transitions sont toutes différentes deux à deux.

Preuve : Soient deux états e_i et e_j du chemin, $i, j \in \mathbb{N}$, $i < j$. Par l'absurde, si $e_i = e_j$ alors le chemin $e_i, t_{i+1}, ..., t_j, e_j$ forme un cycle ce qui est impossible car l'automate est acyclique. Donc tous les états du chemin sont différents. Comme l'automate possède n états, le chemin possède au plus n états, donc il emprunte $n-1$ transitions. □

Chapitre 3

Générateur aléatoire par chaîne de Markov

Dans ce chapitre nous présentons un algorithme de génération aléatoire d'automates acycliques accessibles et déterministes avec une distribution uniforme de la structure de transition. Cet algorithme utilise des résultats sur les chaînes de Markov — appelées aussi processus Markoviens — et sur leurs distributions limite.

Guy Melançon, Isabelle Dutour, et Mireille Bousquet-Mélou ont utilisé ce type de méthode, pour tirer uniformément des graphes acycliques [MDBM01], et Guy Melançon et Fabrice Philippe [MDBM01] l'ont utilisé pour tirer des graphes acycliques connexes. La différence avec notre travail est essentiellement que nous tirons des graphes avec un nombre d'arcs sortants majorés par le cardinal de l'alphabet.

On commence par présenter une première variante de l'algorithme où les états terminaux ne sont pas pris en compte et puis une seconde version où les automates générés sont minimaux. Ces travaux ont été faits avec la collaboration de Vincent Carnino et ont été publiés dans les actes de la conférence CIAA de 2011 [CF11] et dans le jounal TCS — Theoretical Computer Science — pour les automates acycliques minimaux [CF12].

3.1 Algorithme

Dans tout ce chapitre nous considérons uniquement les automates déterministes acycliques et accessibles. Afin de faire court nous les désignerons par le terme raccourci d'≪ automate acyclique ≫.

Notre algorithme consiste à effectuer une suite de déformations aléatoires sur un automate de ce type en préservant ces trois propriétés.

À la suite de chaque déformation l'automate se transforme aléatoirement en un autre automate ≪ proche ≫. Ce processus se modèlise par une chaîne de Markov.

L'algorithme de génération aléatoire prend deux paramètres qui sont : n le nombres d'états de l'automate à produire et T le nombre de de modifications aléatoires à effectuer sur l'automate.

Voici une description sommaire de l'algorithme :

1. On part d'un certain automate à n états qui est déterministe accessible et acyclique. Toujours le même.
2. On effectue une suite de T modifications des transitions de l'automate. Ces modifications sont choisies aléatoirement et peuvent être :

— la suppression d'une transition,
— l'ajout d'une transition,
— la redirection d'une transition,
— on ne change rien.
Après chacune de ces modifications, l'automate reste accessible, déterministe et acyclique.

3. On renvoie l'automate après avoir effectué ces T modifications aléatoires.

La distribution des automates renvoyés par l'algorithme suit un modèle de chaîne de Markov — ou processus de Markov —. Une description plus précise de l'algorithme est proposée à la fin de cette section. Ce qui suit décrit ce qu'est un modèle de chaîne de Markov et rappelle quelques résultats utiles sur ces modèles.

Pour plus de précisions sur les chaînes de Markov le lecteur pourra se référer à [LPW06] ou à [Fel68].

3.1.1 Chaîne de Markov

Une *chaîne de Markov* — ou processus de Markov — est un modèle probabiliste associé à un espace discret et à un temps discret. Dans ce modèle, à chaque instant, la distribution de probabilité sur l'espace dépend uniquement de la distribution à l'instant précédent.

En vérité la définition d'une chaîne de Markov est plus générale, mais nous ne verrons ici que ce type de chaîne que l'on appelle chaîne de Markov homogène.

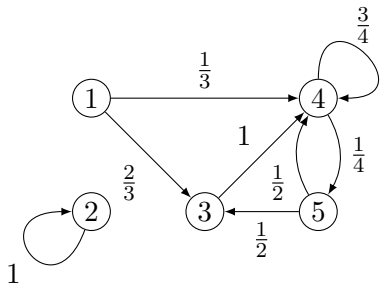
Définition 3 *Une chaîne de Markov dans un espace Ω est un 3-uplet (Ω, p, I) où Ω est un ensemble fini, p une fonction de $\Omega \times \Omega \longrightarrow [0,1]$ et I une fonction de Ω dans $[0,1]$. De plus les fonctions I et p vérifient les deux propriétés suivantes.*
(i) $\forall i \in \Omega$, $\sum_{j\in\Omega} p(i,j) = 1$,
(ii) $\sum_{a\in\Omega} I(a) = 1$.

Les éléments de Ω sont généralement appelés les états de la chaîne de Markov, mais afin de ne pas confondre les états de la chaîne et les états d'un automate, nous disons que Ω est l'ensemble des *places* de la chaîne de Markov.

La fonction I est appelée la *distribution initiale*. Les valeurs $p(i,j)$ sont les *probabilités de transitions*, dans la suite nous noterons $p_{i,j}$ plutôt que $p(i,j)$. Nous avons précisé que l'ensemble espace Ω est fini, ce qui n'est pas toujours le cas des chaînes de Markov. Cependant nous n'étudions ici que des cas finis.

Une chaîne de Markov peut être vue comme un graphe orienté étiqueté, dont les sommets sont les places et où les étiquettes des arcs sont les valeurs des probabilités des transitions. L'étiquette allant de la place i à la place j est donnée par la valeur de $p_{i,j}$. Lorsque cette valeur est nulle, l'arc associé n'est pas dessiné.

Intuitivement, l'étiquette sur un arc allant d'une place j à une place i représente la probabilité pour qu'étant sur la place j à un instant donné u, on ait « sauté » sur la place i entre l'instant u et l'instant $u+1$. Remarquons que rien n'interdit à priori



$$p_{1,3} = \tfrac{2}{3},\ p_{1,4} = \tfrac{1}{3},\ p_{2,2} = 1,\ p_{3,4} = 1,\ p_{4,4} = \tfrac{3}{4}$$

$$p_{4,5} = \tfrac{1}{4},\ p_{5,3} = \tfrac{1}{2},\ p_{5,4} = \tfrac{1}{2},\ p_{1,1} = 0\ p_{1,2} = 0\ p_{1,5} = 0,\ ...$$

FIGURE 3.1 – Une chaîne de Markov à cinq places et sa représentation sous la forme d'un graphe orienté.

que $i = j$. Et comme l'ensemble des places possibles est Ω tout entier, la somme des étiquettes des arcs qui sortent d'une place est toujours égale à 1, ce qui est précisé par la propriété (i) de la définition d'une chaîne de Markov.

L'étude d'une chaîne de Markov consiste principalement à connaître la probabilité d'être à tel ou tel place de la chaîne à un instant donné ou lorsque le temps tend vers l'infini. Comme le temps est discret et infini nous l'associons aux entiers naturels. L'instant initial est l'instant 0, les instants suivants sont les instants 1, puis 2 etc ...

La probabilité d'être à la place i d'une chaîne de Markov à l'instant 0 est donnée par la valeur $I(i)$. Comme Ω représente l'espace tout entier alors la somme des valeurs que prend $I(i)$ lorsque i parcours Ω est égale à 1, ce que précise la propriété (ii) de la définition.

Soit $(X_u)_{u\in\mathbb{N}}$ la suite des distributions successives de la chaîne aux instants $t \in \mathbb{N}$. Chaque distribution X_u est une fonction de Ω dans $[0, 1]$ et $X_u(i)$ exprime la probabilité d'être à la place i à l'instant u.

On a évidemment $X_0 = I$ et, à tout instant u, la somme des $X_u(i)$ lorsque i parcourt Ω est égale à 1. Pour connaître la probabilité d'être à la place j à l'instant $u + 1$ il suffit de connaître la distribution X_u et la probabilité de « sauter » sur la place j lorsqu'on est sur la place i pour tout i. Cette probabilité est donnée par $p_{i,j}$.

La relation entre X_u et X_{u+1} est donc

$$\forall i \in \Omega, X_{u+1}(j) = \sum_{i\in\Omega} X_u(i)p_{i,j}.$$

On remarque que la fonction X_{u+1} dépend entièrement de X_u et des valeurs de $p_{i,j}$ lorsque i et j parcourent Ω.

L'espace Ω est fini, soit ω son cardinal. On associe Ω à l'ensemble $[\omega]$ des entiers naturels, et on fait correspondre les distributions X_u à des vecteurs de taille ω. On

donne à une chaîne de Markov l'interprétation algébrique suivante. Une chaîne de Markov est une suite de vecteurs colonnes $(X_u)_{u\in\mathbb{N}}$ dont les termes sont reliés par la relation de récurrence $X_{u+1} = {}^tPX_u$ où ${}^tP$ est la transposée de la matrice P et où P est une matrice de taille ω et dont les coefficients sont les $p_{i,j}$, le premier indice donnant la ligne et le second la colonne. Le terme initial de cette suite est $I = X_0$.

Ainsi, certaines propriétés algébriques de la matrice P et du vecteur I se traduise par des propriétés sur la chaîne de Markov.

On remarque que pour toute paire d'entiers ℓ, u on a $({}^tP)^\ell X_u = X_{u+\ell}$ et que P^ℓ vérifie la propriété d'une matrice d'une chaîne de Markov et correspond à une transformation de la distribution au bout de ℓ instants.

Nous allons maintenant spécifier quelles sont les conditions suffisantes pour que la suite de distributions d'une chaîne de Markov — X_u dans l'exemple — tende vers la distribution uniforme. Voici pour commencer quelques définitions :

Définition 4 *On dit qu'une chaîne de Markov est* irréductible *si pour tout couple de places i,j il existe un entier ℓ tel que le coefficient i, j de la matrice P^ℓ soit non nul.*

Cela signifie que étant sur n'importe quelle place i de la chaîne on peut atteindre n'importe quel autre place j de la chaîne au bout d'un temps arbitrairement grand. Sur le graphe de la chaîne, cela se traduit par le fait que le graphe est fortement connexe. Cette propriété ne concerne que la fonction p — ou la matrice P — et ne dépend pas de I.

Définition 5 *Le diamètre d'une chaîne de Markov irréductible est le diamètre du graphe sous-jacent.*

Le diamètre de la chaîne est le maximum du plus petit nombre d'arcs qu'il faut emprunter pour aller de n'importe quelle place à n'importe quelle autre place.

Aller d'une place à une autre dans la chaîne peut être réalisé en un nombre d'étapes inférieur ou égal au diamètre.

Définition 6 *On dit qu'une place i d'une chaîne de Markov est* apériodique *si sur le graphe le pgcd des longueurs des cycles passant par i est* 1.

Dit d'une autre façon, i est apériodique si à partir d'un certain temps t_0 la probabilité de retomber sur i en partant initialement de i est non nulle pour tout instant plus grand que t_0.

Comme, sur la matrice P^ℓ, un coefficient $p_{i,j}$ non nul correspond à un chemin de longueur ℓ allant de i vers j, alors cela signifie que le pgcd des éléments de l'ensemble E_i est 1. Où l'ensemble E_i est défini par

$$E_i = \{\ell \in \mathbb{N} / \text{le } i\text{-ème terme de la diagonale de } P^\ell \text{ est non nul}\}.$$

On remarque que si une chaîne est irréductible et possède une place apériodique alors toutes les places sont apériodiques.

Définition 7 *On dit qu'une chaîne de Markov est une* chaîne apériodique. *Si toutes ses places sont apériodiques.*

Définition 8 *On dit qu'une chaîne de Markov est* ergodique *si elle est apériodique et irreductible.*

Définition 9 *Soient deux distributions X_1 et X_2 alors on définit leur distance l'une à l'autre par la formule $||X_1 - X_2|| = \frac{1}{2}\sum_{a\in\Omega}|X_1(a) - X_2(a)|$.*

La définition de cette distance est issue d'une norme d'espace vectoriel réel.

Définition 10 *Soit S une distribution de Ω et soit I la distribution initiale de la chaîne. Si $||({}^tP)^\ell I - S||$ tend vers 0 lorsque ℓ tend vers l'infini on dit que la chaîne tend vers la distribution S. Dans ce cas, il existe deux réels C et α, $\alpha < 1$ tels que $||({}^tP)^\ell I - S|| \leq C\alpha^\ell$.*

On dit que la convergence est exponentielle.

Définition 11 *Une distribution X d'une chaîne de Markov est dite* stationnaire *si ${}^tPX = X$.*

Théorème 5 *Une chaîne de Markov ergodique possède une et une seule distribution stationnaire. De plus pour n'importe quelle distribution initiale I, la chaîne tend vers cette distribution stationnaire.*

Définition 12 *Si une chaîne de Markov est ergodique et si I est sa distribution initiale et S sa distribution stationnaire, on définit le* temps de mélange *d'une chaîne de Markov comme la plus petite valeur d_0 telle que $||({}^tP)^{d_0} I - S|| \leq \frac{1}{4}$.*

Le temps de mélange est une mesure de la vitesse de convergence. En particulier on a $||({}^tP)^{\ell d_0} I - S|| \leq 2^{-\ell}$. Le temps de mélange est en règle général difficile à calculer.

Il existe de nombreuse méthodes permettant d'approcher le temps de mélange d'une chaîne de Markov. Certaines sont très générales et d'autres sont plus spécifiques. Le lecteur peut trouver dans [LPW06] une présentation de ces méthodes.

Théorème 6 *Si la matrice P est symétrique. C'est à dire si pour tout couple de places i, j on a $p_{i,j} = p_{j,i}$ alors la distribution uniforme est une distribution stationnaire.*

Lorsque la matrice P est symétrique on dit aussi que la chaîne de Markov est *symétrique*.

Preuve : Soit U le vecteur de taille ω représentant la distribution uniforme sur Ω. Les coefficients de U sont tous égaux à $\frac{1}{\omega}$. Soient i une place de la chaîne et r_i son coefficient associé sur le vecteur ${}^tPU$. Calculons r_i

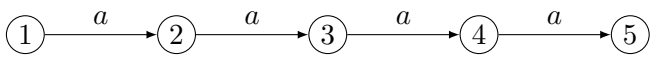


FIGURE 3.2 – Un dessin de $\mathcal{R}_5$

$$\begin{aligned} r_i &= \sum_{j\in\Omega} \tfrac{1}{\omega} p_{j,i}, \\ &= \tfrac{1}{\omega} \sum_{j\in\Omega} p_{i,j} \quad \text{car la matrice est symétrique.} \end{aligned}$$

Or par définition de la matrice P d'une chaîne de Markov, $\sum_{j\in\Omega} p_{i,j} = 1$. Pour toute place i, $r_i = \frac{1}{\omega}$, donc $PU = U$, donc U, la distribution uniforme, est une distribution stationnaire. □

3.1.2 Algorithme

Nous allons maintenant décrire l'algorithme plus en détail.

Une modification sur l'automate est effectuée en fonction d'un triplet (p, x, q) tiré uniformément dans l'ensemble $Q \times A \times Q$. Ce triplet représente la transition allant de l'état p vers l'état q et étiquetée par la lettre x. On note cette transition $p \xrightarrow{x} q$. Si la transition existe alors on la supprime de l'automate. Si l'état p possède une transition étiquetée par x mais qui n'est pas dirigée vers q alors on la redirige vers q. Si l'état p ne possède pas de transition étiquetée par x alors on ajoute à l'automate la transition $p \xrightarrow{x} q$.

On remarque tout de suite que ces opérations préservent la propriété de déterminisme de l'automate.

Après une modification si l'automate reste un automate acyclique — et accessible — on le conserve sinon on annule la modification et on ne change pas l'automate lors de cette étape.

Au début, l'algorithme part toujours d'un même automate que l'on note $\mathcal{R}_n$. C'est un automate acyclique de taille n, dont chaque état de 1 à $n-1$ possède une et une seule transition sortante. Toutes ses transitions ne sont étiquetées que par la lettre a. Son état initial est étiqueté par la lettre 1. Son chemin le plus long en partant de 1 est de longueur $n-1$ — exemple figure 3.2 —.

Pour deux états p, q et une lettre x, on note $\mathcal{A} \ominus p \xrightarrow{x} q$ l'automate $\mathcal{A}$ auquel on a enlevé la transition $p \xrightarrow{x} q$ et $\mathcal{A} \oplus q \xrightarrow{x} p$ l'automate $\mathcal{A}$ auquel on a ajouté la transition $q \xrightarrow{x} p$.

La fonction `EstUnAutomateAcyclique` vérifie si un automate est un automate acyclique, selon la définition que nous avons donnée, c'est à dire si il est déterministe, accessible, et acyclique.

La ligne 10 correspond à l'ajout d'une transition, la ligne 12 à la suppression d'une transition et la ligne 15 à la redirection d'une transition.

L'automate aléatoire renvoyé au bout de T étapes est un automate de taille n acyclique déterministe accessible.

À chaque itération de l'algorithme, l'automate courant ne diffère du précédent que d'au plus une transition. On peut utiliser cette particularité pour rendre l'al-

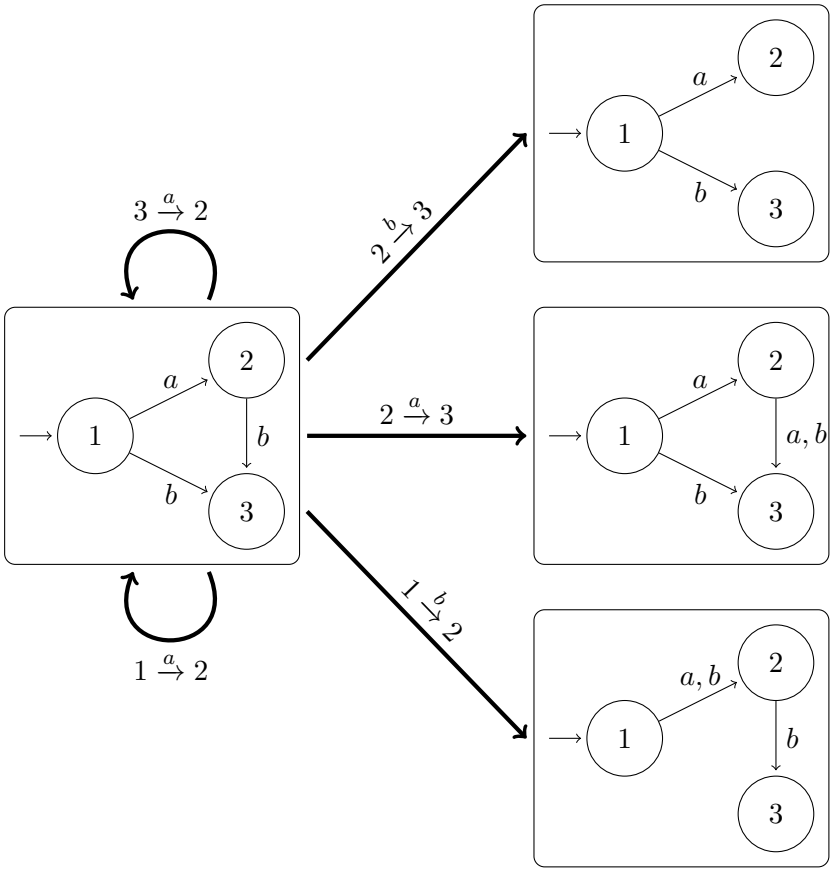


FIGURE 3.3 – Voici une partie de la chaîne de Markov pour $n = 3$. Les places de la chaîne sont des automates à trois états. Ici sont représentées quatre places, et les transitions associées aux triplets tirés.

```
input(n : la taille de l'automate à produire ; T : le nombre de tours de boucle
dans l'algorithme)
output(A : un automate de taille n)
local(R_n : l'automate ligne ; p, q : des états ; a : une lettre)
A ⟵ R_n
for i allant de 1 à T do
    p ⟵ TirerUniformement(Q)
    a ⟵ TirerUniformement(A)
    q ⟵ TirerUniformement(Q)
    if δ(p, a) n'est pas défini then
        if EstUnAutomateAcyclique(A ⊕ p -a-> q) then A ⟵ A ⊕ p -a-> q
    else if δ(p, a) = q then
        if EstUnAutomateAcyclique(A ⊖ p -a-> q) then A ⟵ A ⊖ p -a-> q
    else
        r ⟵ δ(p, a)
        if EstUnAutomateAcyclique(A ⊖ p -a-> r) ⊕ p -a-> q) then
        A ⟵ (A ⊖ p -a-> r) ⊕ p -a-> q
```

gorithme plus efficace en complexité.

Tout d'abord l'ajout, le retrait où la redirection des transitions est faite de façon à conserver la propriété déterministe de l'automate. Il n'est donc jamais nécessaire de vérifier que l'automate $\mathcal{A}$ reste déterministe.

Lorsqu'une transition est retirée nous obtenons un sous-automate du précédent, nous sommes donc certain que l'automate demeure acyclique puisqu'il l'était déjà avant. Lorsqu'on lui ajoute une transition, l'automate reste évidemment accessible.

De plus, nous avons la propriété suivante.

Lemme 14 *Si un automate déterministe est acyclique, alors il est accessible si et seulement si tous les états, excepté l'état initial, possède au moins une transition entrante.*

Preuve : Supposons que l'automate soit accessible, alors pour tout état p il existe un chemin allant de l'état initial, appelons-le 1, à p. La dernière transition qu'emprunte ce chemin est une transition qui aboutie sur l'état p. L'état p possède donc au moins une transition entrante. Supposons maintenant que tout état autre que 1 possède une transition entrante et montrons que l'automate est accessible. Considérons un chemin, le plus long possible, aboutissant sur p. Parce que l'automate est acyclique ce plus long chemin est de longueur finie et les états qu'il traverse sont tous différents. Le point de départ du chemin est un état qui n'a aucune tran-

sition entrante, par l'absurde, s'il en avait une alors le chemin ne serait pas le plus long. Cet état, point de départ du chemin, est nécessairement 1 qui est le seul état sans transition entrante. □

En prenant en compte ces propriétés nous pouvons produire un algorithme semblable au précédent mais qui est plus efficace en complexité.

Algorithm 1: Algorithme optimisé

```
input(n : la taille de l'automate à produire ; T : le nombre de tours de boucle dans l'algorithme)
output(A : un automate de taille n)
local(R_n : l'automate ligne de taille n ; U : un tableau de n nombres entiers ; p, q : des états ; a : une lettre)

A ⟵ R_n
for i allant de 2 à n do U_i = 1

U_1 ⟵ 0
for i allant de 1 à T do
    p ⟵ Uniforme(Q) ; a ⟵ Uniforme(A) ; q ⟵ Uniforme(Q)
    if δ(p, a) n'est pas défini then
        if EstAcyclique(A ⊕ p -a-> q) then
            A ⟵ A ⊕ p -a-> q
            U_q ⟵ U_q + 1
    else if δ(p, a) = q and U_q > 1 then
        A ⟵ A ⊖ p -a-> q
        U_q ⟵ U_q − 1
    else
        r ⟵ δ(p, a)
        if U_r > 1 then
            A ⟵ A ⊖ p -a-> r
            if EstAcyclique(A ⊕ p -a-> q) then
                A ⟵ A ⊕ p -a-> q
                U_q ⟵ U_q + 1
                U_r ⟵ U_r − 1
            else
                A ⟵ A ⊕ p -a-> r
return A
```

La procédure `EstAcyclique` vérifie si le graphe sous-jacent de l'automate est acyclique. Elle peut, par exemple, utiliser l'algorithme classique de parcours de graphe en profondeur qui possède une complexité en $O(n)$.

Théorème 7 *L'algorithme produit un automate acyclique et possède une complexité de $O(nT)$ opérations élémentaires. La chaîne de Markov associée à l'algorithme tend vers la distribution uniforme lorsque le paramètre T tend vers l'infini. La chaîne de Markov possède un diamètre en $O(n)$.*

On rappelle que dans notre définition d'automates déterministes l'état initial est toujours 1.

Bien que l'algorithme ne semble travailler que sur les automates étiquetés, comme il s'agit d'automates accessibles déterministes, il est aussi uniforme sur les automates non étiquetés en vertu du lemme 1 et de la remarque qui le suit.

3.2 Preuves

Cette section montre que la chaîne de Markov associée à l'algorithme tend vers la distribution uniforme. Pour cela on montre que la chaîne est irréductible, ergodique et symétrique.

Preuve (de la complexité) : En utilisant une structure de données adaptée pour représenter l'automate on peut ajouter, rediriger ou supprimer une transition en $O(1)$ opérations. Il existe des algorithmes qui permettent de tester l'acyclicité d'un automate en $O(n)$ opérations et qui sont adaptables à la structure de données utilisée. Chaque accès aux cases du tableau U se fait en $O(1)$ opérations. Tirer uniformément une transition $p \xrightarrow{a} q$ prend $O(1)$ opérations. Donc chaque modification locale peut prendre au plus $O(n+1+1+1) = O(n)$ opérations et comme le nombre de modifications est T alors la complexité totale de l'algorithme est $O(Tn)$. □

Lemme 15 *La chaîne de Markov associée à l'algorithme est symétrique.*

Pour cela nous allons montrer que pour tout couple d'automates $\mathcal{A}$, $\mathcal{B}$, s'il existe un arc allant de $\mathcal{A}$ vers $\mathcal{B}$ alors il existe un arc retour allant de $\mathcal{B}$ vers $\mathcal{A}$ ayant une même probabilité.

Preuve : Regardons un arc de la chaîne de Markov allant d'un automate $\mathcal{A}$ à un automate différent $\mathcal{B}$. Il y a trois cas possibles, soit on enlève à $\mathcal{A}$ une transition $p \xrightarrow{a} q$ pour obtenir $\mathcal{B}$ c'est à dire que $\mathcal{B} = \mathcal{A} \ominus p \xrightarrow{a} q$ soit on ajoute à $\mathcal{A}$ une transition $p \xrightarrow{a} q$, c'est à dire $\mathcal{B} = \mathcal{A} \oplus p \xrightarrow{a} q$, soit on redirige une transition $p \xrightarrow{a} r$ vers un état q, $\mathcal{B} = (\mathcal{A} \ominus p \xrightarrow{a} r) \oplus p \xrightarrow{a} q$. Dans le premier cas si $\mathcal{B} = \mathcal{A} \ominus p \xrightarrow{a} q$ alors $\mathcal{A} = \mathcal{B} \oplus p \xrightarrow{a} q$ ce qui veut dire qu'il existe un arc de la chaîne de $\mathcal{B}$ vers $\mathcal{A}$. Le second cas est le symétrique du premier. Le troisième cas, qui correspond à une redirection, vérifie $\mathcal{B} \ominus p \xrightarrow{a} q = \mathcal{A} \ominus p \xrightarrow{a} r$, et donc $(\mathcal{B} \ominus p \xrightarrow{a} q) \oplus p \xrightarrow{a} r = \mathcal{A}$ et à partir de l'automate $\mathcal{B}$ on obtient l'automate $\mathcal{A}$ en redirigeant la transition $p \xrightarrow{a} q$ vers l'état r. Nous avons montré que pour tout arc allant d'un automate $\mathcal{A}$ à un automate $\mathcal{B}$ — étiqueté par une probabilité non nulle donc — il existe un arc opposé qui va de l'automate $\mathcal{B}$ vers l'automate $\mathcal{A}$. Il faut maintenant montrer que ces deux

arcs ont une même probabilité. Chaque paire d'arcs partant d'un même automate mène à deux automates différents. Le choix de la modification — qui est lié au choix du triplet (p, a, q) — suit une distribution uniforme, en conséquence les arcs de la chaîne de Markov reliant deux automates différents ont tous une probabilité égale à $\frac{1}{kn^2}$ donc, en particulier, deux arcs opposés ont une même probabilité. La chaîne de Markov est donc symétrique. □

Lemme 16 *La chaîne de Markov est apériodique.*

Preuve : Soit $\mathcal{A}$ un automate acyclique quelconque de taille n. La probabilité de tirer la transition $1 \xrightarrow{a} 1$ est non nulle et égale à $\frac{1}{kn^2}$. Or comme le fait de tirer une telle transition conduit l'algorithme à ne pas modifier l'automate, la probabilité de rester sur place est non nulle. Il existe donc, dans la chaîne de Markov, un cycle de longueur 1 qui passe par l'automate $\mathcal{A}$. Ainsi, pour tout automate $\mathcal{A}$, le pgcd de tous les cycles passant par $\mathcal{A}$ est égal à 1. Donc la chaîne de Markov est apériodique. □

Lemme 17 *La chaîne de Markov associée à l'algorithme est irréductible*

Preuve : Pour montrer que la chaîne de Markov est irréductible il faut montrer que pour tout couple d'automates, il y a une probabilité non nulle d'atteindre l'un en partant de l'autre au bout d'un certain temps, c'est à dire qu'il existe un chemin allant de l'un à l'autre et qui emprunte des arcs tous étiquetés par une probabilité non nulle. Pour cela nous allons montrer que pour tout automate $\mathcal{A}$ de taille n il existe un chemin dans la chaîne allant de $\mathcal{A}$ à l'automate ligne $\mathcal{R}_n$. Comme la chaîne de Markov est symétrique — voir lemme 15 — cela implique qu'il existe également un chemin allant de $\mathcal{R}_n$ vers $\mathcal{A}$. Ainsi pour tout couple d'automates acycliques $\mathcal{A}$, $\mathcal{B}$ il existe un chemin allant de $\mathcal{A}$ vers $\mathcal{R}_n$ et il existe un chemin allant de $\mathcal{R}_n$ vers $\mathcal{B}$, donc il existe un chemin allant de $\mathcal{A}$ vers $\mathcal{B}$.

Dans notre cas, pour montrer qu'il existe un chemin allant d'un automate $\mathcal{A}$ à l'automate ligne, il faut montrer que l'on peut appliquer à $\mathcal{A}$ une suite de transformations pour aboutir à l'automate $\mathcal{R}_n$. Ces transformations doivent être celles de l'algorithme, c'est à dire soit l'ajout, soit le retrait, soit la redirection d'une transition. Il faut aussi que, après chaque transformation, l'automate demeure un automate acyclique.

Donc soit $\mathcal{A}$ un automate acyclique, on considère le plus long chemin partant de l'état initial et uniquement composé de transitions étiquetées par la lettre b. On appelle ce chemin C. Ensuite on supprime toutes les transitions aboutissant sur les états du chemin C exceptées celles que C emprunte. L'ordre de ces suppressions n'a pas d'importance. Les automates obtenus par cette suite de transformations sont tous des sous-automates de $\mathcal{A}$, ils sont donc tous acycliques et déterministes.

Appelons $\mathcal{B}$ l'automate obtenu à la fin de cette première suite de transformations. Si nous montrons que $\mathcal{B}$ est accessible, comme $\mathcal{B}$ est un sous-automate de chacun des automates obtenus pour aller de $\mathcal{A}$ à $\mathcal{B}$, alors cela prouvera que tous ces automates sont aussi des automates accessibles.

Soit p un état quelconque de l'automate $\mathcal{B}$, comme il est accessible dans l'automate de départ $\mathcal{A}$, il y existe un chemin allant de l'état initial vers p. Si ce chemin n'emprunte aucune des transitions retirées alors il est conservé dans $\mathcal{B}$ et l'état p est accessible dans $\mathcal{B}$. Si ce chemin emprunte des transitions retirées il passe forcément par un des états de C. Soit r le dernier état de C par lequel il passe. La partie du chemin allant de r à p existe dans $\mathcal{B}$ puisque qu'elle ne traverse aucun état de C, de plus r est accessible puisque c'est un état du chemin C et que C part de l'état initial. Donc il existe un chemin allant de l'état initial à r et il existe un chemin allant de r à p. Donc l'état p est accessible.

Maintenant il existe deux cas, dans le premier, le nombre d'états du chemin C dans l'automate $\mathcal{B}$ est n, dans le second ce nombre est inférieur à n. Si nous sommes dans le second cas, nous allons « faire grandir » C jusqu'à être dans le premier cas.

Soit q le dernier état du chemin C et soit s un état quelconque qui ne soit pas dans C, ajoutons à $\mathcal{B}$ une transition qui part de q, qui aboutit à s et qui est étiquetée par la lettre b. Nous obtenons un nouvel automate $\mathcal{B}'$, $\mathcal{B}'$ est évidement accessible montrons qu'il est déterministe et acyclique. C a été choisi comme le plus long chemin empruntant uniquement des transitions étiquetées par la lettre b. Ainsi q ne possède donc aucune transition sortante étiquetée par b dans l'automate $\mathcal{A}$. Dans le cas contraire q ne serait pas le dernier état de C. L'automate $\mathcal{B}$ est un sous-automate de $\mathcal{A}$ donc q ne possède aucune transition étiquetée par b dans l'automate $\mathcal{B}$. Donc en ajoutant une transition à q on ne rompt pas la propriété déterministe et donc $\mathcal{B}'$ est déterministe.

Montrons que $\mathcal{B}'$ est acyclique par l'absurde. Si il existe un cycle dans $\mathcal{B}'$, puisque $\mathcal{B}$ ne possède aucun cycle, un cycle de $\mathcal{B}'$ passe nécessairement par la transition rajoutée $q \xrightarrow{b} s$. Il existe donc dans l'automate $\mathcal{B}$ un chemin allant de l'état s à l'état q. Comme q est un état de C et que s n'est pas dans C il existe une transition qui part d'un état n'appartenant pas à C pour aboutir à un état de C cette transition n'appartient donc pas à C si on se réfère à la façon dont le chemin C est construit. C'est impossible puisque dans l'automate $\mathcal{B}$ toutes les transitions qui aboutissent à un état du chemin C appartiennent au chemin C. Donc l'hypothèse d'un cycle dans $\mathcal{B}'$ ne tient pas, $\mathcal{B}'$ est acyclique.

Dans l'automate $\mathcal{B}'$, on a augmenté la longueur du chemin C, le chemin partant de l'état initial et qui emprunte uniquement des transitions étiquetées par la lettre b est strictement plus grand que celui de l'automate $\mathcal{A}$ qui est un automate quelconque. De plus l'étiquette du dernier état s de cette chaîne a été choisie arbitrairement parmi les états qui n'appartiennent pas à C. Nous avons atteint l'automate $\mathcal{B}'$ en partant de $\mathcal{A}$ et en lui faisant subir des modifications locales, qui correspondent à des transitions de la chaîne de Markov. On recommence le processus pour atteindre un automate uniquement composé de transitions étiquetées par la lettre b.

Ensuite on recommence le procédé mais cette fois ci en ajoutant uniquement des transitions étiquetées par la lettre a et en choisissant les sommets sur lesquelles aboutissent ces transitions dans l'ordre de leurs étiquettes. L'automate obtenu après

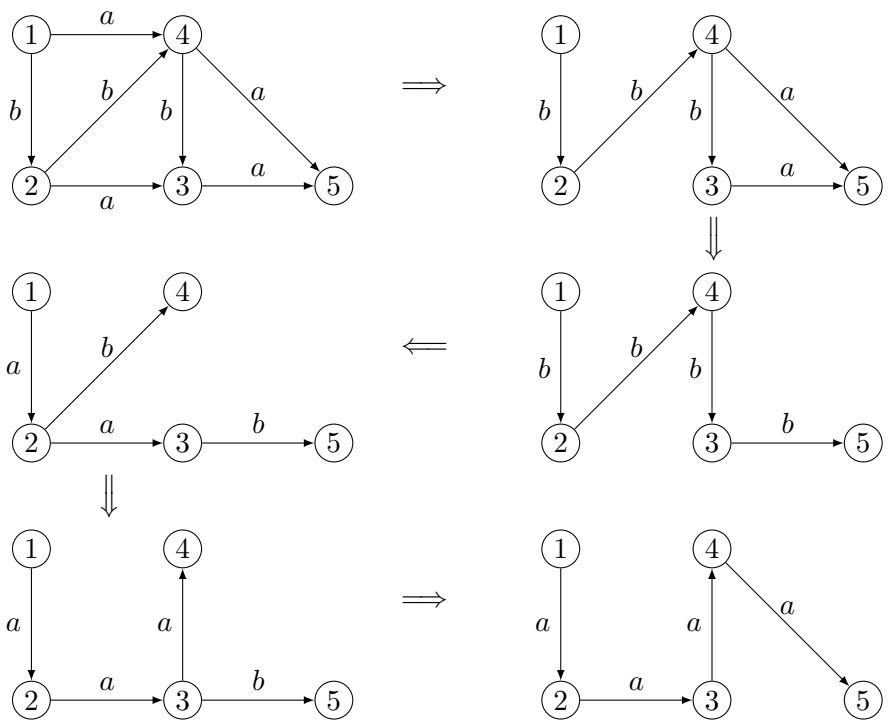


FIGURE 3.4 – Exemple d'une suite d'automates par lesquels passe l'algorithme pour transformer un automate en l'automate $\mathcal{R}_5$

ces nouvelles modifications est l'automate ligne $\mathcal{R}_n$. □

Cette preuve est une preuve constructive qui décrit la suite des états par lesquels on peut passer pour relier n'importe quel automate à l'automate $\mathcal{R}_n$. En comptant le double du nombre d'automates de cette suite, on obtient une borne supérieure du diamètre de la chaîne.

Lemme 18 *Le diamètre de la chaîne de Markov est en $O(n)$.*

Preuve : La première étape de la preuve qui consiste à produire le plus grand chemin étiqueté par la lettre b se fait en un nombre d'opérations borné supérieurement par $n(k+1)$, car il faut retirer à chaque état au plus k transitions puis lui ajouter une transition b. Ensuite la reconstruction en l'automate $\mathcal{R}_n$ nécessite $2(n-1)$ opérations puisque on retire $n-1$ transitions b et on ajoute $n-1$ transitions a. □

Preuve (du théorème 7) : On a montré que la chaîne de Markov associée à l'algorithme est apériodique, symétrique, et irréductible. Elle tend donc vers une probabilité uniforme. □

3.3 Algorithme sur automates acycliques minimaux

L'algorithme précédent ne prend pas en compte les états terminaux des automates générés. La suite de ce chapitre présente une autre version de l'algorithme

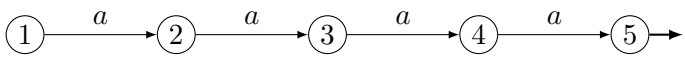


FIGURE 3.5 – Un dessin de $\mathcal{D}_5$

qui génère aléatoirement et uniformément un automate acyclique minimal.

Définition 13 *On appelle automate acyclique minimal un automate qui est acyclique, déterministe, accessible et minimal.*

Le fonctionnement de l'algorithme est semblable a celui du précédent, on effectue T modifications locales, choisies aléatoirement, sur un automate et qui conservent un automate acyclique minimal.

L'automate de départ est l'automate $\mathcal{D}_n$ qui est semblable à l'automate $\mathcal{R}_n$. C'est un automate acyclique minimal en forme de ligne qui possède un seul état initial 1 et un seul état terminal : l'état n — voir figure 3.5 —.

1. On part d'un automate toujours le même $\mathcal{D}_n$.
2. On lui fait subir une suite de T modifications aléatoires locales qui conservent sa propriété d'automate acyclique minimal. Ces modifications peuvent être :
   — la transformation d'un état terminal en état non terminal ou alors la transformation inverse — transformer un état non terminal en état terminale —,
   — la redirection d'une transition qui existe déjà,
   — la suppression d'une transition qui existe déjà,
   — l'ajout d'une transition qui n'existait pas,
   — on ne fait rien.
3. On renvoie l'automate obtenu au bout des T modifications.

On présente en page 65 une description détaillée de l'algorithme. On utilise les notations introduites lors de la description du premier algorithme dans la section précédente. On ajoute la notation $\mathcal{A} \oplus \bar{p}$, $p \in [n]$, qui représente l'automate $\mathcal{A}$ où on a transformé l'état p en état terminal (respectivement non terminal) s'il était non terminal (respectivement terminal).

L'algorithme tire à pile ou face s'il va modifier, soit la structure de transition de l'automate, soit l'ensemble des états terminaux. Ce dernier cas est simplement effectué en ajoutant ou en retirant à l'ensemble des états terminaux un état tiré aléatoirement. La modification de la structure de transition est effectuée de façon similaire à l'algorithme précédent. On ajoute des procédures de vérification dans les cas où l'automate est suceptible de n'être plus minimal, ni acyclique. Dans cette situation-là on annule la dernière modification jusqu'au prochain tour de boucle.

Le tableau V permet de compter le nombre de transitions sortant d'un état. On verra plus tard que pour qu'un automate acyclique soit minimal il ne doit posséder qu'un seul état sans transition sortante et que cet état doit être un état terminal.

Algorithm 2: Markov minimal

input(n : *la taille de l'automate à produire ;* T : *le nombre de tours de boucle dans l'algorithme*)
output($\mathcal{A}$: *un automate de taille* n)
local(U, V : *deux tableaux* n *de nombres entiers ;* $\mathcal{D}_n$: *l' automate ligne*)

$\mathcal{A} \longleftarrow \mathcal{D}_n$
for p *allant de* 2 *à* n **do** $U_p = 1$
for p *allant de* 1 *à* $n-1$ **do** $V_p = 1$

$U_1 \longleftarrow 0$; $V_n \longleftarrow 0$
for i *allant de* 1 *à* T **do**
  $b \longleftarrow$ `Uniforme`$(\{0,1\})$
  if $b = 1$ **then**
    $p \longleftarrow$ `Uniforme`(Q) ; $a \longleftarrow$ `Uniforme(A)` ; $q \longleftarrow$ `Uniforme(Q)`
    if $\delta(p,a)$ *n'est pas défini* **then**
      if `EstAcyclique`$(\mathcal{A} \oplus p \xrightarrow{a} q)$ **and** `EstMinimal`$(\mathcal{A} \oplus p \xrightarrow{a} q)$ **then**
        $\mathcal{A} \longleftarrow \mathcal{A} \oplus p \xrightarrow{a} q$
        $U_q \longleftarrow U_q + 1$
        $V_p \longleftarrow V_p + 1$
    else if $\delta(p,a) = q$ **and** $U_q > 1$ **and** $V_p > 1$ **and**
    `EstMinimal`$(A \ominus p \xrightarrow{a} q)$ **then**
      $\mathcal{A} \longleftarrow \mathcal{A} \ominus p \xrightarrow{a} q$
      $U_q \longleftarrow U_q - 1$
      $V_p \longleftarrow V_p - 1$
    else
      $r \longleftarrow \delta(p,a)$
      if $U_r > 1$ **then**
        $\mathcal{A} \longleftarrow \mathcal{A} \ominus p \xrightarrow{a} r$
        if `EstAcyclique`$(\mathcal{A} \oplus p \xrightarrow{a} q)$ **and** `EstMinimal`$(\mathcal{A} \oplus p \xrightarrow{a} q)$ **then**
          $\mathcal{A} \longleftarrow \mathcal{A} \oplus p \xrightarrow{a} q$
          $U_q \longleftarrow U_q + 1$
          $U_r \longleftarrow U_r - 1$
        else
          $\mathcal{A} \longleftarrow \mathcal{A} \oplus p \xrightarrow{a} r$
else
  $p \longleftarrow$ `Uniforme`$([\mathrm{n}])$
  if `EstMinimal`$(\mathcal{A} \oplus \bar{p})$ **then**
    $\mathcal{A} \longleftarrow \mathcal{A} \oplus \bar{p}$
return $\mathcal{A}$

Pour vérifier qu'un automate est minimal — fonction `EstMinimal` — on peut utiliser l'algorithme de Dominique Revuz [Rev92] qui s'applique aux automates acycliques et qui possède une complexité linéaire.

Théorème 8 *La complexité de cet algorithme est $O(nT)$. La chaîne de Markov associée a l'algorithme tend vers la distribution uniforme. Le diamètre de la chaîne de Markov est en $O(n)$.*

3.4 Preuves

Avant montrer le théorème nous allons présenter et démontrer quelques propriétés utiles sur les automates acycliques minimaux.

On introduit une classe d'automates déterministes acycliques : les automates en hamac.

Définition 14 *Soit $\mathcal{A}$ un automate. On dira que $\mathcal{A}$ est un automate en hamac si*
- *Il est déterministe et acyclique.*
- *Pour tout état i de $\mathcal{A}$, il existe un chemin allant de 1 à n passant par i.*

Lemme 19 *Soit $\mathcal{A}$ un automate acyclique — selon notre définition c'est à dire acyclique déterministe et accessible —. Alors $\mathcal{A}$ est un automate en hamac si et seulement si pour chaque état i, $i \neq n$, i possède au moins une transition sortante.*

Preuve : Soit un état i, comme $\mathcal{A}$ est accessible il existe un chemin allant de 1 à i. Soit C un des plus long chemin passant par i. Ce chemin est de longueur finie. Soit t l'état d'arrivé du chemin. Montrons que $t = n$ par l'absurde, si $t \neq n$ alors il existe au moins une transition sortant de t. Soit q l'état vers lequel est dirigée cette transition. Si on ajoute l'état q à la fin du chemin C on obtient un chemin plus long. Impossible par hypothèse sur C, donc l'hypothèse $t \neq n$ ne tient pas. Donc il existe un chemin allant de i à n. L'autre sens est facile. □

Définition 15 *Soit $\mathcal{A}$ un automate sans cycle. Soit un état p et soit C un des plus long chemin partant de p. La longueur de C est appelée le* rang *de i dans $\mathcal{A}$.*

Dans un automate en hamac le rang d'un état p est la longueur de l'un des plus long chemin partant de p et atteignant n. Un automate acyclique est en hamac si et seulement si n est le seul état de rang 0. Si l'automate est émondé alors le rang d'un état est le maximum des longueurs des mots reconnaissables par cet état. On note rang(i) le rang d'un état i. On peut voir, sur la figure 3.6, un exemple des rangs des états d'un automate.

Définition 16 *Soient $\mathcal{A} = (Q, A, \delta, F)$ un automate déterministe, et deux états p et q. On dit que p et q sont* directement équivalents *si ils vérifient les deux propriétés suivantes :*

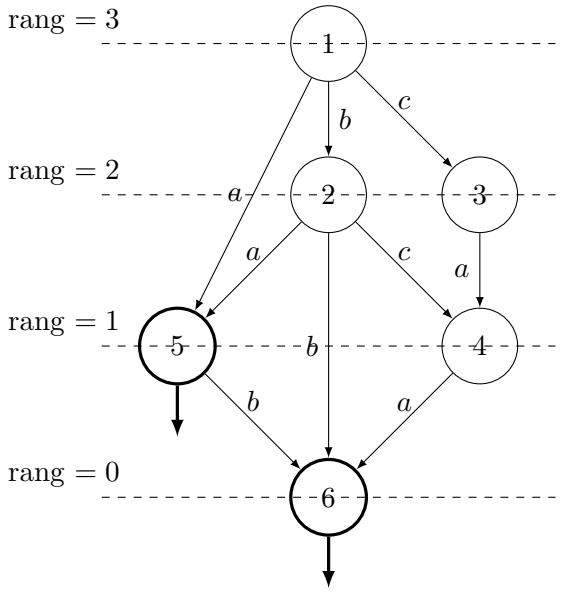


FIGURE 3.6 – Dessin d'un automate acyclique où les états possédant un même rang sont reliés par un trait en pointillés.

— *p est un état terminal si et seulement si q est un état terminal,*
— *$A_q = A_p$ et pour toute lettre x, $x \in A_q$ on a $\delta(p, x) = \delta(q, x)$.*

Dans n'importe quel automate, deux états directement équivalents sont équivalents. Nous allons voir que la réciproque est vrai lorsque l'automate est en hamac.

Définition 17 *Soient p et q deux états d'un automate acyclique. S'il existe un chemin allant de p à q on dit que p est un* ancêtre *de q.*

Le lemme suivant énumère quelques propriétés que l'on utilisera dans la suite. Ces propriétés, assez évidentes, ne sont pas démontrées dans ce manuscrit.

Lemme 20 *Soit $\mathcal{A}$ un automate acyclique — et déterministe et accessible — et p et q deux états de $\mathcal{A}$, on a les propriétés suivantes.*

i) Si p est un ancêtre de q et $p \neq q$ alors $\mathrm{rang}(q) < \mathrm{rang}(p)$.
ii) Si p est un état de rang k alors pour tous k entre 0 et k il existe un état de rang i dans $\mathcal{A}$.
iii) Si $\mathcal{A}$ est émondé et si p et q sont équivalents alors $\mathrm{rang}(p) = \mathrm{rang}(q)$.
iv) Si le seul état de rang 0 — sans transition — est n alors l'automate $\mathcal{A}$ est un automate en hamac.

Lemme 21 *Soit un automate $\mathcal{A}$ tel que l'état n ne possède pas de transition sortante, les deux propositions suivantes sont équivalentes :*

— *$\mathcal{A}$ est un automate acyclique minimal,*
— *$\mathcal{A}$ est un automate en hamac, il ne possède pas deux états directement équivalents et l'état n est un état terminal.*

Preuve : $\Rightarrow$ Supposons que $\mathcal{A}$ soit un automate acyclique minimal. Soit q un état sans transition sortante. Montrons par l'absurde que q est forcément terminal. Supposons q non terminal, l'ensemble des mots reconnaissables par l'état q est vide. L'automate $\mathcal{A}$ privé de l'état q reconnaît le même langage que $\mathcal{A}$, donc $\mathcal{A}$ n'est pas minimal.

Donc q est terminal et comme n ne possède aucune transition sortante alors n est aussi terminal. Le seul mot que reconnaît q est le mot vide, de même le seul mot que reconnaît n est le mot vide donc n et q sont équivalents et donc comme $\mathcal{A}$ est minimal $q = n$. Donc le seul état sans transition est n et il est terminal.

Comme $\mathcal{A}$ est accessible et que tout état, excepté n, possède une transition sortante alors $\mathcal{A}$ est un automate en hamac d'après le lemme 19. Puisque $\mathcal{A}$ est minimal il ne possède pas deux états directement équivalents.

$\Leftarrow$ Supposons maintenant que $\mathcal{A}$ soit un automate en hamac, qu'il ne possède pas deux états directement équivalents et que son état n soit terminal. D'après la définition d'un automate en hamac, $\mathcal{A}$ est déterministe, il ne possède aucun cycle et tous ses états sont accessibles depuis l'état initial. De plus pour tout état i il existe un chemin allant de i à n qui est un état terminal. L'automate $\mathcal{A}$ est donc émondé.

Montrons qu'il ne possède pas deux états différents équivalents. Par l'absurde, soient r, s deux états différents et équivalents tel que le rang de r soit le plus petit possible. Comme $\mathcal{A}$ est émondé et que r et s sont équivalents alors $\text{rang}(r) = \text{rang}(s)$. De plus r est terminal si et seulement si s est terminal, $A_r = A_s$ et pour toute lettre x de A_r, $\delta(r, x)$ est équivalent à $\delta(s, x)$. Mais comme $\forall x, x \in A_p$, $\text{rang}(\delta(r, x)) = \text{rang}(\delta(s, x)) < \text{rang}(r)$ et que par hypothèse sur r il n'existe pas d'état de rang inférieur qui soit équivalent à un autre état alors on a $\forall x, x \in A_p$, $\delta(r, x) = \delta(s, x)$ c'est à dire que s et r sont directement équivalents. Impossible sur $\mathcal{A}$ donc $r = s$. □

Dorénavant nous conviendrons que le seul état sans transition sortante d'un automate acyclique minimal est n.

Preuve (Complexité) : La modification d'une transition de l'automate ou la modification qui consiste à changer un état terminal en état non terminal ou le contraire, se fait en un nombre d'opérations constant. Il existe des algorithmes pour vérifier l'acyclicité d'un automate en $O(n)$ opérations que l'on peut utiliser pour la fonction `EstAcyclique`. L'automate étant acyclique, on peut vérifier que l'automate est minimal en un nombre d'opérations linéaire en n grâce à l'algorithme de Revuz [Rev92]. Donc chaque modification locale peut prendre au plus $O(n)$ opérations et comme le nombre de modifications est T alors la complexité totale de l'algorithme est $O(Tn)$. □

Lemme 22 *La chaîne de Markov associée à l'algorithme est ergodique.*

Preuve : Il y a une probabilité non nulle pour qu'un automate acyclique minimal ne soit pas modifié lors d'un tour de l'algorithme, par exemple, lorsque le tirage au sort mène à la construction d'un cycle dans l'automate. Cela signifie que dans la

chaîne de Markov il existe un cycle de longueur 1 sur cet automate et donc que la chaîne est apériodique.

Les modifications locales de l'algorithme, qui peuvent être la modification d'une transition précise ou la modification de l'ensemble des états terminaux mènent chacune à des automates différents. Chacune de ces modifications peut être annulée par une modification réciproque qui a la même probabilité d'être exécutée. La chaîne de Markov est donc symétrique.

La chaîne étant apériodique et acyclique, elle est ergodique. □

Lemme 23 *La chaîne de Markov associée à l'algorithme est irréductible.*

La preuve procède en deux temps. D'abord on montre que à partir de n'importe quel automate $\mathcal{A}$ on peut atteindre un automate isomorphe à l'automate $\mathcal{D}_n$. Ensuite que l'on peut permuter des états de l'automate tout en conservant la propriété d'automate acyclique minimal afin d'atteindre $\mathcal{D}_n$. La symétrie de la chaîne permet de conclure.

Preuve : Nous allons montrer que à partir de tout automate acyclique minimal $\mathcal{A}$ nous pouvons atteindre l'automate $\mathcal{D}_n$ en utilisant uniquement des modifications décrites dans l'algorithme. Comme la chaîne de Markov est symétrique cela veut dire que de $\mathcal{D}_n$ nous pouvons atteindre n'importe quel autre automate acyclique minimal. Donc que le graphe associé à la chaîne de Markov est fortement connexe et donc que la chaîne de Markov est irréductible.

Soit un automate acyclique minimal $\mathcal{A}$. Posons t le rang de l'état 1 qui est l'état initial. Il y a deux cas, dans le premier : $t = n-1$, dans le second : $t < n-1$. Si $t < n-1$ on utilise le lemme 24 pour atteindre un automate qui possède un état initial de rang égal à $n-1$.

Sans changer de notation supposons donc que l'état initial de $\mathcal{A}$ possède un rang égal à $n-1$. Comme l'automate est accessible alors chaque état de l'automate possède un rang compris entre 0 et $n-1$. De plus comme il existe un état de rang $n-1$ alors pour chaque rang compris entre 0 et $n-1$ il existe un état qui possède ce rang. Donc tous les états de $\mathcal{A}$ possèdent des rangs différents.

Nous appelons $e_0, e_1, ..., e_{n-2}, e_{n-1}$ les états de l'automate tel que $\text{rang}(e_i) = i$. Par exemple si $\mathcal{A} = \mathcal{D}_n$, on a $e_i = n - i$.

Pour chaque état e_i, $i > 0$, il existe au moins une transition allant de l'état e_i à l'état e_{i-1}. Pour chaque valeur de rang possible i, $i > 0$, on redirige toutes les transitions partant de l'état e_i vers l'état e_{i-1} et on ajoute à l'état e_i les transitions non définies que l'on dirige vers l'état e_{i-1}.

On remarque que chaque transition ajoutée ou redirigée a été dirigée vers un état de rang inférieur, en conséquence l'automate reste acyclique, ainsi les rangs des états ne changent pas et sont tous différents donc l'automate reste minimal.

Comme les rangs des états ne changent pas, le seul état avec un rang égal à 0 reste n. Comme pour tout état e_i, $i < n-1$, e_i possède au moins une transition

entrante issue de l'état e_{i+1} alors l'automate reste un automate accessible donc en hamac.

Ensuite en partant de l'automate obtenu, on supprime toutes les transitions non étiquetées par la lettre a. Après chacune de ces suppressions l'automate reste évidemment acyclique et déterministe. Comme, après chaque suppression, pour chaque $i > 0$ il existe au moins une transition allant de l'état e_i vers l'état e_{i-1} alors les rangs des états sont conservés et tout état, excepté e_{n-1}, possède une transition entrante donc l'automate reste en hamac et minimal. Ensuite on modifie l'automate de façon à rendre tous les états non terminal, excepté l'état n. Ça ne change pas les rangs des états, l'automate reste un automate acyclique minimal.

L'automate obtenu, que l'on appelle désormais $\mathcal{B}$, est un automate isomorphe à $\mathcal{D}_n$. Nous allons maintenant « permuter » les rangs des états de cet automate en fonction leurs étiquettes de façon à atteindre l'automate $\mathcal{D}_n$.

Pour cela nous allons montrer que pour tout couple d'entier i, k ,$0 < i < i+k < n-1$ il existe une suite de modifications sur l'automate qui produit un automate isomorphe à $\mathcal{B}$ mais qui modifie les rangs des états selon la permutation cyclique $(i+k, i+k-1, i+k-2, .., i)$ tout en conservant à chaque étape la propriété d'automate acyclique minimal. Un exemple de ces transformations est donné dans la figure 3.7 qui insère l'état 3 entre l'état 2 et 5. Ces permutations cycliques engendrent l'ensemble de toutes les permutations ayant pour support $\{1, \ldots, n-2\}$ ce qui permet d'ordonner les états de façon à obtenir l'automate $\mathcal{D}_n$. La suite de la preuve décrit cette suite d'opérations et montre que chacune préserve la propriété d'automate acyclique minimal.

On ajoute à e_{i+k+1} une transition b dirigée vers l'état de rang inférieur e_{i+k}. L'automate reste acyclique, les rangs des états sont conservés. On redirige la transition $e_{i+k+1} \xrightarrow{a} e_{i+k}$ vers e_i. L'état e_{i+k} reste accessible par la transition ajoutée juste avant, les rangs des états ne changent pas. On rend e_i terminal. Les rangs des états ne changent pas. On redirige la transition $e_{i+1} \xrightarrow{a} e_i$ vers e_{i-1}. L'état e_j reste accessible par la transition redirigée juste avant, l'automate reste acyclique, déterministe et en hamac. Les états e_i et e_{i+1} sont de même rang et donc susceptibles d'être directement équivalents ils ne le sont pas car e_i est terminal et pas e_{i+1}. On redirige la transition $e_i \xrightarrow{a} e_{i-1}$ vers e_{i+k}. L'automate reste acyclique car la seule transition arrivant sur l'état e_i provient de l'état e_{i+k+1} qui possède un rang plus grand que celui de e_{i+k}. On rend l'état e_i non terminal. Les états possèdent tous des rangs différents. On supprime la transition $e_{i+k+1} \xrightarrow{b} e_{i+k}$. L'automate est maintenant de nouveau isomorphe à l'automate $\mathcal{D}_n$, et on a permuté les rangs des états selon la permutation cyclique $(i+k, i+k-1, i+k-2, .., i)$. □

Si on compte le nombre d'opérations que décrit la preuve on peut alors majorer le diamètre de la chaîne. Pour obtenir un automate où l'état 1 est de rang $n-1$ il faut effectuer au plus $n-1$ ajouts de transitions. Ensuite on retire certaines transitions pour obtenir un automate isomorphe à $\mathcal{D}_n$. Pour cela on effectue au plus $k(n-1)$ opérations. L'opération de permutation cyclique des rangs des états qui est décrite à la fin de la preuve se fait en un nombre constant d'opérations. Pour

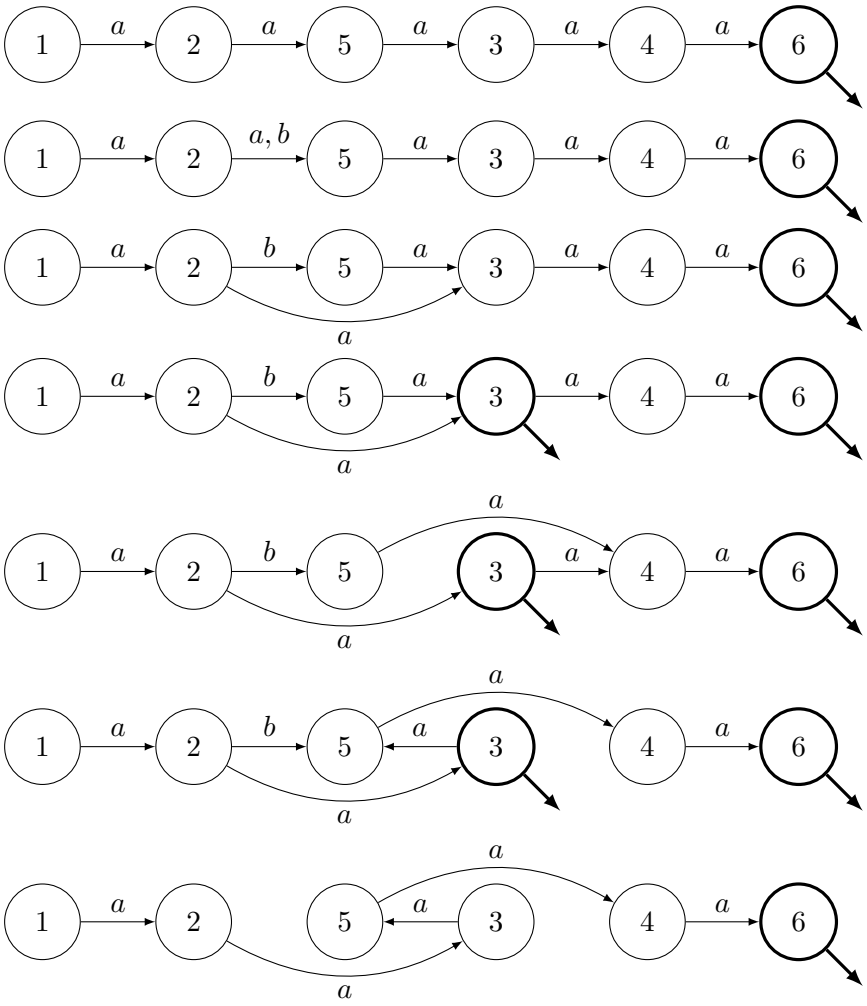


FIGURE 3.7 – Insertion de l'état 3 après l'état 2. Ici la permutation est (5,3). À chaque étape on conserve un automate acyclique minimal.

ordonner correctement les états de l'automate il faut au plus effectuer $n-3$ de ces permutations si on les choisit bien. On peut en déduire le lemme suivant.

Corollaire 1 *La chaîne de Markov possède un diamètre en $O(n)$.*

Le lemme suivant est utilisé dans la preuve précédente.

Lemme 24 *Pour tout automate acyclique minimal $\mathcal{A}$ de taille n, il existe un chemin dans la chaîne de Markov de l'algorithme qui permet à partir de $\mathcal{A}$ d'atteindre un automate où l'état initial possède un rang égal à $n-1$.*

Preuve : Supposons que l'état initial soit de rang k, avec k inférieur strictement à $n-1$. Nous allons montrer que, en redirigeant une transition bien choisie, on peut augmenter strictement le rang de l'état initial et conserver un automate acyclique minimal. Ainsi en redirigeant au plus $n-1-k$ transitions on peut atteindre un automate acyclique minimal qui possède un état initial de rang égal à $n-1$.

Il y a n états dans $\mathcal{A}$, tous de rangs compris entre 0 et $n-1$, donc il existe deux états ayant le même rang. Choisissons p et q deux état différents de même rang, le plus petit possible. Puisque $\mathcal{A}$ est en hamac le seul état de rang 0 est n et donc $\text{rang}(p) = \text{rang}(q) > 0$. Soit un état t et une lettre x tel que la transition $p \xrightarrow{x} t$ existe et que $\text{rang}(t)+1 = \text{rang}(q)$. Alors comme q et p ont le même rang il existe un état t' et une lettre x' tel que la transition $p \xrightarrow{x'} t'$ existe et que $\text{rang}(t')+1 = \text{rang}(q)$. On en déduit que $\text{rang}(t') = \text{rang}(t)$ or comme $\text{rang}(t) < \text{rang}(p)$ et que p et q ont le même rang, le plus petit possible alors $t = t'$. Soit un état r qui possède le plus grand rang parmi les états qui ne sont pas des ancêtres de p. Un tel r existe car l'ensemble des états qui ne sont pas ancêtres de p n'est pas vide, q en fait partie. On a $\text{rang}(p) = \text{rang}(q) \leq \text{rang}(r)$. On redirige la transition $p \xrightarrow{x} t$ vers l'état r. On obtient un nouvel automate, appelons-le l'automate $\mathcal{B}$.

La suite montre que l'automate $\mathcal{B}$ est un automate acyclique minimal dont le rang de l'état initial est strictement supérieur à k.

L'automate $\mathcal{B}$ est acyclique car r n'est pas un ancêtre de p dans $\mathcal{A}$. Puisque l'automate $\mathcal{B}$ est acyclique on peut définir le rang d'un état dans $\mathcal{B}$, nous noterons $\text{rang}_{\mathcal{A}}(i)$ et $\text{rang}_{\mathcal{B}}(i)$ le rang d'un état i dans l'automate $\mathcal{A}$ et dans l'automate $\mathcal{B}$ respectivement. Remarquons qu'un état est un ancêtre de p dans l'automate $\mathcal{A}$ si et seulement si c'est ancêtre de p dans l'automate $\mathcal{B}$ et que si un état i n'est pas un ancêtre de p alors $\text{rang}_{\mathcal{B}}(i) = \text{rang}_{\mathcal{A}}(i)$. Puisque $\text{rang}_{\mathcal{A}}(p) \leq \text{rang}_{\mathcal{A}}(r)$ et qu'il existe une transition $p \to r$ dans l'automate $\mathcal{B}$ alors $\text{rang}_{\mathcal{B}}(p) > \text{rang}_{\mathcal{B}}(r) = \text{rang}_{\mathcal{A}}(r)$.

L'état t reste accessible par la transition $q \xrightarrow{x'} t$ et comme le nombre de transitions sortant de chaque état dans $\mathcal{B}$ est le même que dans $\mathcal{A}$ alors selon la propriété *iv)* du lemme 20 l'automate $\mathcal{B}$ est en hamac. L'automate $\mathcal{B}$ est évidemment déterministe.

Montrons que $\mathcal{B}$ est minimal. Par l'absurde, si l'automate $\mathcal{B}$ n'est pas minimal alors selon le lemme 21 il existe deux états directement équivalents. Comme

l'automate $\mathcal{A}$ est minimal et que la seule différence entre l'automate $\mathcal{A}$ et l'automate $\mathcal{B}$ est la direction de la transition $p \xrightarrow{x} \cdot$, alors il existe un état p' directement équivalent à p. Puisque p et p' sont directement équivalents alors p' n'est pas un ancêtre de p dans $\mathcal{B}$, donc p' n'est pas un ancêtre de p dans $\mathcal{A}$ et donc $\text{rang}_{\mathcal{A}}(p') = \text{rang}_{\mathcal{B}}(p')$. Comme p et p' sont équivalents dans $\mathcal{B}$ ils ont le même rang et donc $\text{rang}_{\mathcal{A}}(r) = \text{rang}_{\mathcal{B}}(r) < \text{rang}_{\mathcal{B}}(p) = \text{rang}_{\mathcal{B}}(p') = \text{rang}_{\mathcal{A}}(p')$. Or r a été choisi parmi les états qui ne sont pas des ancêtres de p dans $\mathcal{A}$ et ayant le rang le plus grand possible et justement p' n'est pas un ancêtre, c'est en contradiction avec $\text{rang}_{\mathcal{A}}(r) < \text{rang}_{\mathcal{A}}(p')$. L'hypothèse $\mathcal{B}$ n'est pas minimal ne tient pas, donc $\mathcal{B}$ est un automate acyclique minimal.

Montrons que, après cette redirection, le nouveau rang de l'état initial est strictement supérieur à k. Par l'absurde supposons que $\text{rang}_{\mathcal{B}}(1) \leq \text{rang}_{\mathcal{A}}(1)$. Alors l'ensemble des ancêtres i de p tels que $\text{rang}_{\mathcal{B}}(i) \leq \text{rang}_{\mathcal{A}}(i)$ n'est pas vide. Soit u un élément de cet ensemble tel que son rang dans $\mathcal{A}$ soit le plus petit possible. On a $u \neq p$ car $\text{rang}_{\mathcal{A}}(p) < \text{rang}_{\mathcal{B}}(p)$. Soit un état v et une lettre y tel que $u \xrightarrow{y} v$ et que $\text{rang}_{\mathcal{A}}(v) + 1 = \text{rang}_{\mathcal{A}}(u)$. Deux cas se présentent, dans un cas v est un ancêtre de p, dans l'autre il ne l'est pas.

• Si v est un ancêtre de p alors $\text{rang}_{\mathcal{A}}(v) + 1 \leq \text{rang}_{\mathcal{B}}(v)$ puisque $\text{rang}_{\mathcal{A}}(v) < \text{rang}_{\mathcal{A}}(u)$. Donc $\text{rang}_{\mathcal{A}}(u) \leq \text{rang}_{\mathcal{B}}(v)$ or comme u est un ancêtre de v dans $\mathcal{B}$ alors $\text{rang}_{\mathcal{A}}(u) \leq \text{rang}_{\mathcal{B}}(v) < \text{rang}_{\mathcal{B}}(u)$. C'est impossible par hypothèse sur u.

On regarde donc le second cas.

• Si v n'est pas un ancêtre de p alors $\text{rang}_{\mathcal{B}}(v) = \text{rang}_{\mathcal{A}}(v) \leq \text{rang}_{\mathcal{A}}(r) = \text{rang}_{\mathcal{B}}(r)$. Donc $\text{rang}_{\mathcal{B}}(v) \leq \text{rang}_{\mathcal{B}}(r) < \text{rang}_{\mathcal{B}}(r) + 1 = \text{rang}_{\mathcal{B}}(p) < \text{rang}_{\mathcal{B}}(u)$ car $u \neq p$, ainsi $\text{rang}_{\mathcal{B}}(v) + 2 \leq \text{rang}_{\mathcal{B}}(u)$. On déduit de cette dernière inéquation que $\text{rang}_{\mathcal{A}}(v) + 2 \leq \text{rang}_{\mathcal{B}}(u)$, mais on a aussi que $\text{rang}_{\mathcal{A}}(v) + 1 = \text{rang}_{\mathcal{A}}(u)$, donc $\text{rang}_{\mathcal{A}}(u) + 1 \leq \text{rang}_{\mathcal{B}}(u)$, ainsi $\text{rang}_{\mathcal{A}}(u) < \text{rang}_{\mathcal{B}}(u)$ impossible par hypothèse sur u.

Les deux cas que nous avons regardés mènent tous les deux à une absurdité. L'hypothèse $\text{rang}_{\mathcal{B}}(1) \leq \text{rang}_{\mathcal{A}}(1)$ ne tient pas, donc $\text{rang}_{\mathcal{A}}(1) < \text{rang}_{\mathcal{B}}(1)$. □

3.5 Conclusion

Les algorithmes de générations aléatoires basés sur les chaînes de Markov sont souvent faciles à implanter et garantissent qu'à la limite on atteint la distribution souhaitée. Ils sont également souples au sens où l'on peut facilement ajouter des contraintes tant que l'on s'assure que la chaîne reste irréductible.

En revanche, quantifier le temps de mélange peut s'avérer extrêmement difficile. Dans ce chapitre nous en avons vu l'illustration : nos algorithmes sont aisés à concevoir mais on ne sait pas vraiment quand les arrêter.

La méthode de couplage par le passé, lorsqu'elle s'applique, permet de s'affranchir du temps de mélange puisqu'elle garantie que les productions de l'algorithme suivent la distribution stationnaire. [PW96]

L'incertitude sur le paramètre de mélange de l'algorithme se traduit par l'incer-

titude du temps d'exécution. Malheureusement nous n'avons pas réussi à appliquer cette technique pour les automates acycliques.

Une piste pour améliorer l'algorithme serait d'accélérer le test d'acyclicité. En effet il est coûteux de reproduire un test complet alors que nous savons que seule une transition a été modifiée, nous n'avons cependant pas trouvé d'optimisation qui soit significative comparée au test naïf. La même question se pose au sujet du test de minimalité.

Nous nous sommes donc intéressé à une autre méthode qui utilise la décomposition récursive. Elle est présentée dans le chapitre suivant.

Chapitre 4

Générateur aléatoire par la méthode récursive

Dans ce chapitre, nous décrivons un algorithme de génération aléatoire uniforme d'automates acycliques. Cet algorithme utilise une méthode de type récursif et possède une complexité en $O(n^3)$. Nous verrons en fin de chapitre que cette complexité est même meilleur en pratique.

L'algorithme prend en entrée un nombre entier n et produit un automate, de taille n, tiré uniformément parmi tous les automates acycliques de taille n.

La méthode récursive est une technique classique de génération aléatoire — voir [NW75],[FZC94] —. Elle utilise une décomposition combinatoire de l'objet en objets ayant des tailles plus petites. Si on peut calculer le nombre d'objets d'une taille donnée — par exemple grâce à la décomposition elle-même — on peut, par un procédé naturel, en déduire un algorithme de génération aléatoire selon une distribution uniforme.

La première partie présente des résultats de décompositions d'automates acycliques suivis par une première version du générateur qui repose naturellement sur cette décomposition. La seconde partie présente un algorithme plus efficace et affiné grâce à des résultats typiques sur les automates acycliques. L'algorithme de cette seconde partie est plus détaillé.

Ce travail, fait en collaboration avec Cyril Nicaud a fait l'objet d'une publication dans la conférence CSR [FN13b].

4.1 Algorithme général

4.1.1 Décomposition d'automates acycliques

Nous rappelons qu'on appelle « automate acyclique » un automate déterministe qui possède un graphe sous-jacent ne comportant aucun cycle. Contrairement au chapitre précédent les automates que nous traitons ici ne sont pas forcément accessibles et peuvent donc posséder plusieurs états sources, par contre ils seront toujours déterministes. Les automates sont étiquetés et les états terminaux et initiaux ne sont pas considérés.

Pour faciliter les preuves et les descriptions d'automates nous travaillons souvent sur les complétés des automates. Dans ce contexte l'état puits est noté $\{\perp\}$, l'état puits ne compte pas dans la taille de l'automate, δ est une véritable application définie pour tout couple (q, a) où q parcours l'ensemble des états et a l'ensemble

des lettres de A. Les transitions partant de l'état puits sont toutes dirigées vers l'état puits.

Lorsqu'on représente un automate acyclique par une figure on omet évidemment les transitions dirigées vers l'état puits. De plus elles ne sont pas prises en compte pour définir la propriété d'acyclicité.

L'automate de taille 0 ou *automate vide* est l'automate dont le complété possède un unique état puits.

Définition 18 *Soit un triplet $\mathcal{A} = ([n], A, \delta)$ où A est un alphabet et δ une application de $[n] \times A$ dans $[n] \cup \{\perp\}$. Si le graphe sous-jacent de l'application δ est acyclique alors $\mathcal{A}$ est appelé un automate acyclique. On note $\mathbb{A}$ l'ensemble de ces automates. Pour un entier n on note $\mathbb{A}_n$ l'ensemble des automates acycliques de taille n.*

Un automate acyclique non vide possède au moins un état sans transition entrante que l'on appelle une source ou un état source de l'automate. Pour s'en convaincre il suffit de partir d'un état quelconque et d'emprunter à l'envers une des transitions qui atteignent cet état puis de recommencer ce procédé tant que c'est possible. Parce que l'automate est sans cycle, il n'est pas possible, lors de ce cheminement, de rencontrer un état déjà atteint auparavant. Le nombre d'états de l'automate étant fini, le procédé s'arrête nécessairement et le dernier état atteint est une source.

Nous aurons souvent besoin de parler du nombre de sources d'un automate, pour cela on définit la fonction src.

Définition 19 *Pour un automate $\mathcal{A}$ on note* $\mathrm{src}(\mathcal{A})$ *son nombre de sources.*

Définition 20 *On pose η l'application qui enlève les sources d'un automate et normalise les étiquettes des états restants.*

D'un autre point de vue l'application η est une procédure qui choisi un sous-automate normalisé d'un automate $\mathcal{A}$ de façon déterministe.

L'application η est à priori définie sur n'importe quel type d'automates étiquetés, pas forcément un automate acyclique.

L'image d'un automate par η peut éventuellement être l'automate vide.

Lemme 25 *On a les propriétés suivantes :*

- *L'image d'un automate par η est un sous-automate normalisé de son antécédent.*
- *L'application η envoie un automate déterministe acyclique non vide sur un autre automate déterministe acyclique de taille strictement plus petite.*

Preuve : Un sous-automate d'un automate acyclique est également acyclique. Les transitions enlevées sont celles qui ont pour origine les états enlevés donc l'automate reste également déterministe complet. Si un automate est acyclique et non vide alors il possède au moins une source, ainsi son image par η est un automate plus petit. □

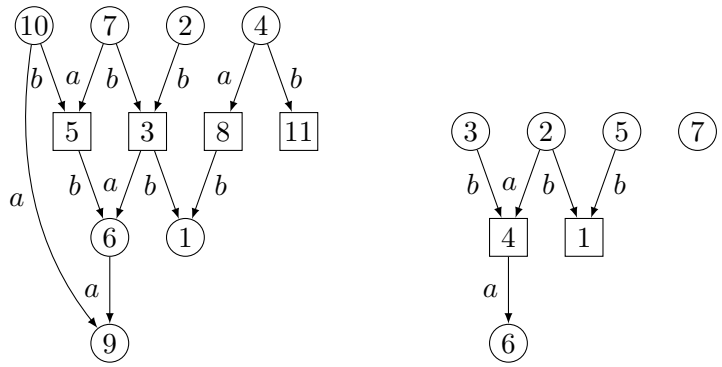


FIGURE 4.1 – Deux automates acycliques. Celui de droite est l'image de celui de gauche par l'application η. Les sources secondaires de chaque automate sont carrées.

Pour un automate $\mathcal{A}$, on appelle l'ensemble des *sources secondaires* de $\mathcal{A}$ les états correspondant aux sources de l'image de $\mathcal{A}$ par η.

Nous avons mis en évidence une décomposition récursive d'automates acycliques. Cette décomposition va nous permettre de compter le nombre d'automates acycliques et de concevoir un algorithme de génération aléatoire uniforme.

La transformation qui consiste à supprimer les sources et à renommer l'automate est bien une application, on montre facilement que c'est une surjection. Cependant ce n'est pas une injection. L'image réciproque d'un automate par cette application est un ensemble infini.

Définition 21 *Pour un automate $\mathcal{R}$ on note $\mathcal{M}_n(\mathcal{R})$ l'ensemble des automates acycliques $\mathcal{A}$ de taille n qui vérifient $\eta(\mathcal{A}) = \mathcal{R}$.*

Une autre façon de définir $\mathcal{M}_n(\mathcal{R})$: $\mathcal{M}_n(\mathcal{R}) = \eta^{-1}(\mathcal{R}) \cap \mathbb{A}_n$.

Le lemme qui suit parle d'une caractérisation des automates de $\mathcal{M}_n(\mathcal{R})$ et va nous permettre de calculer son cardinal.

Lemme 26 *Soient $\mathcal{R}$ et $\mathcal{A}$ deux automates acycliques — et déterministes —. On considère leurs complétés. On pose S l'ensemble des sources de $\mathcal{A}$ et on pose R le complémentaire de S, état puits inclus. $\mathcal{A}$ appartient à $\mathcal{M}_n(\mathcal{R})$ si et seulement si les propriétés suivantes sont toutes vérifiées :*

(i) La taille de $\mathcal{A}$ est n.

(ii) $\mathcal{R}$ est un sous-automate normalisé de $\mathcal{A}$ — état puits inclus — et ses états correspondant aux états de $\mathcal{R}$ sont les états de l'ensemble R.

Preuve : Si $\mathcal{A}$ appartient à $\mathcal{M}_n(\mathcal{R})$ alors il est par définition de taille n. On rappelle que l'état puits n'est pas compté dans la taille. Si $\eta(\mathcal{A}) = \mathcal{R}$ alors par définition de l'application η, $\mathcal{R}$ est un sous-automate normalisé de $\mathcal{A}$. Les états de S sont supprimés par l'application η, les états de R sont donc les états correspondant aux états de $\mathcal{R}$.

La réciproque est également quasi-immédiate. □

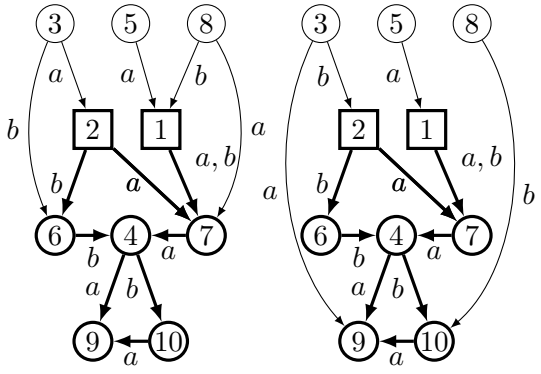


FIGURE 4.2 – Deux automates acycliques de taille 10 qui ont la même image par l'application η. Les sources secondaires des deux automates sont représentées par des carrés.

Lemme 27 *Soit un automate déterministe complet pas forcément acyclique $\mathcal{A}$, $\mathcal{A} = (n, A, \delta)$. Soit S un sous-ensemble d'états $[n]$ de $\mathcal{A}$. On appelle R le complémentaire de S.*

Alors S est l'ensemble des sources de $\mathcal{A}$ si et seulement si on a les trois propriétés suivantes :

(a) Toutes les transitions sortant des états de S sont dirigées vers des états de R — i.e. $\delta(S, A) \subset R$ —.

(b) Toutes les transitions sortant des états de R sont dirigées vers les états de R — i.e. $\delta(R, A) \subset R$ —.

(c) Tout état de R possède au moins une transition entrante — i.e. $R \subset \delta([n], A)$ —.

Preuve : Un état est une source si et seulement si il ne possède aucune transition entrante. Si l'ensemble S est exactement l'ensemble des sources alors tout autre état est atteint par une source, on a donc $\delta([n], A) = [n] \setminus S = R$. Donc on a (c) directement et comme $S \subset [n]$ et $R \subset [n]$ alors on a (a) et (b). Supposons maintenant que l'on ait (a), (b) et (c). On en déduit tout de suite $\delta([n], A) = R$. □

Lemme 28 *Soit $\mathcal{R}$ un automate de taille r. Le cardinal de $\mathcal{M}_n(\mathcal{R})$ dépend uniquement de n, r et de u le nombre de sources de $\mathcal{R}$.*

On a $|\mathcal{M}_n(\mathcal{R})| = \binom{n}{r}\beta(n, u, r)$, où $\beta(n, u, r)$ est un entier.

Intuitivement, la valeur $\beta(n, u, r)$ est le nombre de façons de relier $n-r$ nouveaux états aux r états de $\mathcal{R}$ de manière à ce que chaque source de $\mathcal{R}$ ait au moins une transition entrante. C'est cette contrainte qui nous permet de garantir que tous les états correspondant aux états de $\mathcal{R}$ ne sont pas supprimés lors de l'application η.

On donne une formule explicite des valeurs $\beta(n, u, r)$ plus loin dans le lemme 29.

Preuve : Calculons le cardinal de $|\mathcal{M}_n(\mathcal{R})|$. Nous nous appuyons pour cela sur le lemme 26 et ses notations : on travaille sur les complétés des automates acycliques

considérés ; pour un automate de $\mathcal{M}_n(\mathcal{R})$ on pose S l'ensemble de ses sources et R les états restants.

On partitionne $\mathcal{M}_n(\mathcal{R})$ selon les étiquettes correspondant aux états de R. La condition *(ii)* du lemme 26 indique que R est nécessairement de cardinal r. Comme les automates de $\mathcal{M}_n(\mathcal{R})$ possèdent n états il y a $\binom{n}{r}$ parties dans cette partition. On travaille maintenant dans l'une de ces parties, on suppose donc que les ensembles S et R sont fixés. On va calculer le cardinal de l'une de ces parties. En s'appuyant sur la définition 18 d'un automate acyclique, compter les éléments d'une partie consiste à dénombrer le nombre de fonctions de transition δ qui vérifient les deux propriétés du lemme 26. Selon le lemme 26, l'automate $\mathcal{R}$ — état puits inclus — est un sous-automate normalisé de $\mathcal{A}$, ainsi, les destinations des transitions sortant des états R sont fixées. En conséquence, la fonction de transition δ restreinte aux états de R est la même pour deux automates pris dans une même partie de la partition. Les seules différences possibles entre deux automates d'une même partie sont donc les valeurs prises par δ sur l'ensemble $S \times A$. On pose U l'ensemble des états de R correspondant aux sources de $\mathcal{R}$. On va montrer que la proposition *(ii)* du lemme 26 est vérifiée si et seulement si

$$U \subset \delta(S, A) \subset R. \tag{4.1}$$

Pour cela on va utiliser le lemme 27 qui permet de caractériser l'ensemble des sources d'un automate par les propriétés (a),(b) et (c). Avant de commencer la démonstration, nous remarquons que — U étant l'ensemble des sources de $\mathcal{R}$ — $\delta(R, A) = R \setminus U$.

Partons de la formule 4.1. On a directement $\delta(S, A) \subset R$ donc la propriété (a) est vérifiée. Les états R correspondent au sous-automate $\mathcal{R}$ complet donc la proposition (b) est vérifiée.

On a également

$$\begin{aligned}
\delta([n], A) &= \delta(S \cup R, A),\\
\delta([n], A) &= \delta(S, A) \cup \delta(R, A),\\
\delta([n], A) &= \delta(S, A) \cup R \setminus U.
\end{aligned}$$

Or $U \subset \delta(S, A)$ donc $R \subset \delta(S, A) \cup R \setminus U = \delta([n], A)$, (c) est donc vérifiée.

Maintenant partons des propriétés (a), (b) et (c) et montrons la formule 4.1. La propriété (a) implique directement $\delta(S, A) \subset R$. On a

$$\begin{aligned}
\delta(S, A) &= \delta([n] \setminus R, A),\\
\delta(S, A) &\supset \delta([n], A) \setminus \delta(R, A).
\end{aligned}$$

Or on sait que $\delta(R, A) = R \setminus U$ et selon (c) on a $\delta([n], A) \supset R$ donc

$$\begin{aligned}
\delta(S, A) &\supset R \setminus (R \setminus U),\\
\delta(S, A) &\supset U.
\end{aligned}$$

Donc le cardinal d'une partie est donnée par le nombre de fonctions δ vérifiant 4.1 sur $S \times A$. On note ce cardinal $\beta(n, r, u)$, il dépend uniquement des cardinaux des ensembles S, R, U et du cardinal de l'alphabet A. Le cardinal $|S|$ dépend de r et n — car $|S| = n - r$ —. Pour ces raisons, toutes les parties de la partition ont un même cardinal qui est $\beta(n, r, u)$.

Comme nous avons montré qu'il y a $\binom{n}{r}$ parties, la somme des cardinaux de toutes les parties est $\binom{n}{r}\beta(n, u, r) = |\mathcal{M}_n(\mathcal{R})|$. □

Dans le lemme précédent, nous avons introduit les valeurs $\beta(n, u, r)$, dans le lemme qui suit, nous donnons une formule explicite de ces valeurs.

Lemme 29 *Soient les valeurs entières n, u, r, $u \leq r$, $r \leq n$ et les ensembles S, R et U tels que $U \subset R$. On suppose que les ensembles S, R et U sont de cardinal respectifs $n - r$, $r + 1$ et u. Alors $\beta(n, u, r)$ compte le nombre de fonctions δ allant de $S \times A$ vers R, telles que $U \subset \delta(S, A)$.*

On a

$$\beta(n, u, r) = \sum_{i \in [k(n-r)]} \binom{k(n-r)}{i} \mathrm{Surj}(i, u) \cdot (r - u + 1)^{k(n-r)-i},$$

où $\mathrm{Surj}(i, u)$ *compte le nombre total de surjections différentes qui partent d'un ensemble de cardinal i vers un ensemble de cardinal u. Les coefficients* Surj *vérifient*

$$\begin{aligned}
\mathrm{Surj}(i, u) =& u \cdot \mathrm{Surj}(i-1, u-1) + u \cdot \mathrm{Surj}(i-1, u) && \text{si } 1 \leq u \leq i,\\
\mathrm{Surj}(i, 1) =& 1 && \text{si } 1 \leq i,\\
\mathrm{Surj}(i, u) =& 0 && \text{si } i < u.
\end{aligned}$$

Preuve : On partitionne l'ensemble recherché en fonction du nombre d'éléments de $S \times A$ qui ont leurs images dans U par l'application δ. On pose i ce nombre. Pour un i donné, on choisit les i transitions qui aboutissent sur les états de U ce qui fait $\binom{k(n-r)}{i}$ ensembles possibles. Ensuite la restriction de l'application δ à l'un de ces ensembles est une surjection qui prend ses valeurs dans U. Les images des autres éléments de $S \times A$ sont incluses dans $R \setminus U$ sans aucune contrainte. On a donc $\beta(n, u, r) = \sum_{i \in [k(n-r)]} \binom{k(n-r)}{i} \mathrm{Surj}(i, u) \cdot (r - u + 1)^{k(n-r)-i}$. □

Définition 22 *On pose $\alpha(n)$ le nombre d'automates de taille n. On pose $\alpha(n, s)$ le nombre d'automates de taille n possédant s sources.*

Lemme 30 *Pour tout couple d'entiers naturels n, s, $s \leq n$, on a les deux égalités suivantes :*

i) $\alpha(n) = \sum_{s=1}^{n} \alpha(n, s)$.

ii) $\alpha(n, s) = \sum_{u=1}^{n} \binom{n}{s} \beta(n, u, n-s)\alpha(n-s, u)$.

Preuve : *i)* On partitionne l'ensemble des automates acycliques de taille n en fonction du nombre de sources.

ii) On rappelle que $\mathrm{src}(\mathcal{A})$ compte le nombre de source d'un automate $\mathcal{A}$.

On partitionne les automates acycliques en fonction de leurs images par η. Puisque les automates considérés possèdent s sources alors par η ils sont tous forcément de taille $n-s$.

Pour un entier s on a

$$\mathbb{A}_n \bigcap \mathrm{src}^{-1}(\{s\}) = \bigsqcup_{\mathcal{R}\in\mathbb{A}_{n-s}} \mathcal{M}_n(\mathcal{R}).$$

On partitionne à nouveau en fonction du nombre de sources secondaires de $\mathcal{R}$. On a

$$\mathbb{A}_n \bigcap \mathrm{src}^{-1}(\{s\}) = \bigsqcup_{u=1}^{n} \bigsqcup_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} \mathcal{M}_n(\mathcal{R}),$$

$$\alpha(n,s) = \sum_{u=1}^{n} \sum_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} |\mathcal{M}_n(\mathcal{R})|.$$

Puisque l'on sait que le cardinal de $\mathcal{M}_n(\mathcal{R})$ ne dépend que de la taille et du nombre de sources de $\mathcal{R}$ — cf. lemme 28 — et que $\mathrm{src}(\mathcal{R}) = u$, on peut remplacer $|\mathcal{M}_n(\mathcal{R})|$ dans la formule. On obtient

$$\begin{aligned}
\alpha(n,s) &= \sum_{u=1}^{n} \sum_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} \binom{n}{n-s} \beta(n,u,n-s),\\
&= \sum_{u=1}^{n} \binom{n}{n-s} \beta(n,u,n-s) \sum_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} 1,\\
&= \sum_{u=1}^{n} \binom{n}{s} \beta(n,u,n-s)\alpha(n-s,u).
\end{aligned}$$

□

Lemme 31 *Pour n, s, u entiers, $s < n$ et $u \leq n-s$, le nombre d'automates acycliques à n états, s sources et u sources secondaires est donné par*

$$\binom{n}{s} \beta(n,u,n-s)\alpha(n-s,u).$$

Preuve : Soit $\mathbb{B}_{n,s,u}$ l'ensemble qui nous interresse. On le partitionne selon les images obtenus par l'application η.

On a

$$\begin{aligned}\mathbb{B}_{n,s,u} &= \bigsqcup_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} \{\eta(\mathcal{A})=\mathcal{R} | \mathcal{A}\in\mathbb{A}_n\},\\ &= \sum_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} |\{\eta(\mathcal{A})=\mathcal{R} | \mathcal{A}\in\mathbb{A}_n\}|.\end{aligned}$$

Selon le lemme 28, le cardinal de l'image réciproque d'un automate acyclique par η dépend uniquement de sa taille et de son nombre de sources, donc

$$\begin{aligned}|\mathbb{B}_{n,s,u}| &= \binom{n}{s}\beta(n,u,n-s)\sum_{\substack{\mathcal{R}\in\mathbb{A}_{n-s}\\ \mathrm{src}(\mathcal{R})=u}} 1,\\ &= \binom{n}{s}\beta(n,u,n-s)\cdot|\mathbb{A}_{n-s}\cap \mathrm{src}^{-1}(\{u\})|,\\ &= \binom{n}{s}\beta(n,u,n-s)\cdot\alpha(n-s,u).\end{aligned}$$

□

Nous avons maintenant assez de résultats pour définir l'algorithme récursif de génération aléatoire uniforme. On présente d'abord un algorithme simple ; le nombre d'états et le nombre de sources sont des paramètres. À la fin de la section suivante, nous décrivons par un pseudo-code un autre algorithme qui produit les mêmes objets mais qui est plus efficace en complexité.

4.1.2 Algorithme

Voici un algorithme qui permet de tirer uniformément un automate acyclique complété de taille n possédant s états sources.

1- On tire u, le nombre de sources secondaires de cet automate.

2- On tire un automate acyclique $\mathcal{R}$ de taille $n-s$ ayant u sources en utilisant récursivement l'algorithme.

3- On tire S, $S\subset[n]$, l'ensemble des étiquettes des s états sources puis on ré-étiquette les états de $\mathcal{R}$ par les états de $[n]\setminus S$ en conservant l'ordre.

4- On tire les transitions allant des états S vers l'automate $\mathcal{R}$ de façon que ses u sources soient atteintes.

Chacune des étapes de l'algorithme choisit aléatoirement une structure. Il faut garantir que les distributions associées à ces 4 choix conservent l'uniformité de la distribution des automates produits par l'algorithme.

On peut voir le déroulement de l'algorithme sur un exemple dessiné sur la figure 4.3.

Les paragraphes qui suivent donnent les détails de chacunes de ces 4 étapes.

Étape 0 : on pose les 3 sources.

Étape 1 : on tire le nombre de sources secondaires (ici 2).

Étape 2 : on tire un automate acyclique de taille $10 - 3 = 7$ ayant 2 sources.

Étape 3 : on tire les étiquettes des trois sources (ici {3,5,8}) et on renumérote les autres états.

Étape 4 : on tire les transitions provenant des sources.

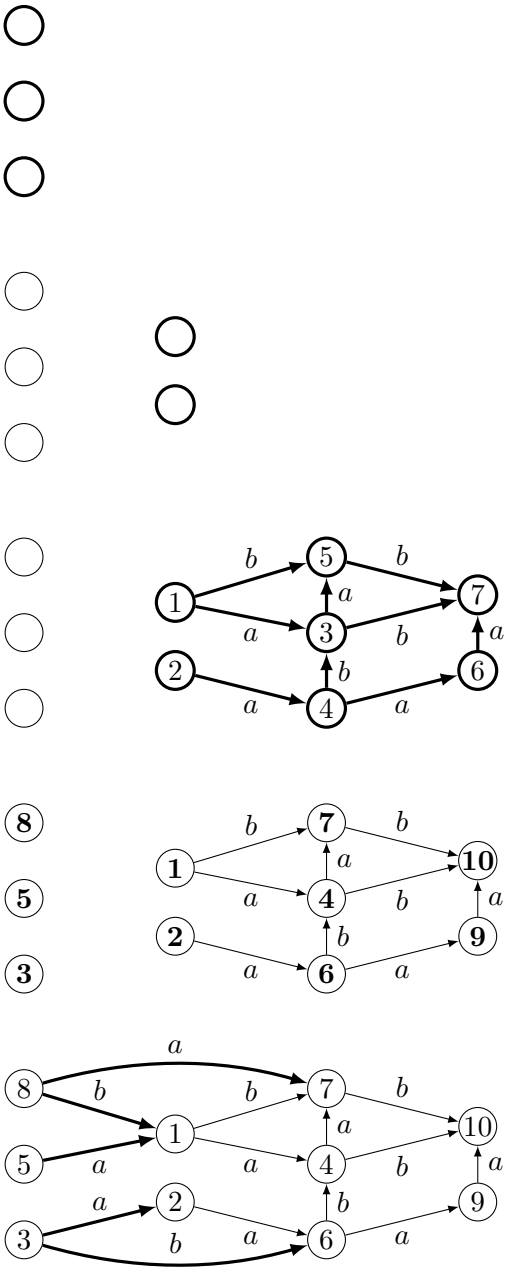


FIGURE 4.3 – Le déroulement des 4 étapes de l'algorithme sur un exemple de taille 10 ($n = 10$) et ayant 3 sources ($s = 3$). On montre l'automate avant et après chaque étape.

1) Le lemme 31 nous permet de calculer la probabilité qu'un automate de taille n ayant s sources, possède u sources secondaires. Cette probabilité est donnée par $\frac{\binom{n}{s}\beta(n,s,n-u)\alpha(n-s,u)}{\alpha(n,s)}$. Pour tirer le nombre de sources secondaires d'un automate uniformément il suffit de partitionner l'intervalle entier $[\alpha(n,s)]$ en $n-s$ intervalles I_u chacun de cardinal $\binom{n}{s}\beta(n,s,n-u)\alpha(n-s,u)$, où $1 \leq u \leq n-s$ puis de tirer uniformément un entier dans $[\alpha(n,s)]$ et de retourner l'indice de l'intervalle dans lequel se trouve l'entier tiré.
2) Il s'agit d'appliquer l'algorithme avec des paramètres différents. Lorsque le paramètre n prend une valeur nulle alors l'algorithme doit renvoyer l'automate vide — l'automate de taille 0 —.
3) L'ensemble des parties de cardinal s parmi l'ensemble $[n]$ mène à la décomposition bien connu suivante : le plus grand élément parmi les s entiers est soit n soit un des $n-1$ autres entiers. Le cardinal du premier ensemble est $\binom{n-1}{s-1}$ le cardinal du second ensemble est $\binom{n-1}{s}$. Le nombre d'éléments est au total $\binom{n}{s} = \binom{n-1}{s} + \binom{n-1}{s-1}$. La probabilité que n appartienne à l'ensemble est donc $\frac{\binom{n-1}{s}}{\binom{n}{s}}$. La probabilité que n n'appartienne pas à l'ensemble est donc $1 - \frac{\binom{n-1}{s}}{\binom{n}{s}}$. On utilise cette décomposition pour chaque élément de $[n]$ afin de tirer s éléments parmi n.
4) Tous les états sources de l'automate $\mathcal{R}$ doivent être atteints par des transitions provenant de S. On peut décomposer cette répartition en fonction de i qui est le nombre de transitions dirigées vers les états sources de $\mathcal{R}$.
   Les états sources sont différentiés car étiquetés et chaque transition possède une étiquette dans l'alphabet A. Les transitions provenant des états de S sont donc toutes différentiées. Le choix de l'ensemble des i transitions atteignant les états sources de $\mathcal{R}$ parmi les ks transitions partant de S s'apparente au choix classique d'un ensemble de i éléments parmi ks. Pour ce choix on peut adapter l'algorithme de l'étape 3). Le nombre de choix est $\binom{ks}{i}$.
   La répartition de ces transitions peut être vue comme le nombre de surjections d'un ensemble i vers u le nombre de sources de $\mathcal{R}$. On utilise la décomposition classique d'une surjection : l'image du plus grand élément e possède soit un, soit plusieurs antécédents. Le nombre de surjections correspondant au premier cas est $u \cdot \mathrm{Surj}(i-1, u-1)$. Le nombre de surjection du second cas est $u \cdot \mathrm{Surj}(i-1, u)$. On a donc

$$\mathrm{Surj}(i,u) = u \cdot \big(\mathrm{Surj}(i-1,u-1) + \mathrm{Surj}\ (i-1,u)\big).$$

   Les cas triviaux limites sont $\mathrm{Surj}(0,u) = 0$ et $\mathrm{Surj}(i,1) = 1$. La probabilité que l'image de e ait un unique antécédent est $\frac{u \cdot \mathrm{Surj}(i-1,u-1)}{\mathrm{Surj}(i,u)}$. La probabilité que l'image de e ait plus d'un antécédent est $\frac{u \cdot \mathrm{Surj}(i-1,u)}{\mathrm{Surj}(i,u)}$. L'image de e est choisie selon une distribution uniforme sur $[u]$.
   Les transitions qui ne sont pas dirigées vers les sources de $\mathcal{R}$ sont distribuées vers les $n-s-u+1$ autres états et indépendamment les une

des autres. Ainsi la probabilité que l'une de ces transitions atteigne un de ces états est $\frac{1}{n-s-u+1}$. Le nombre de ces transitions est $ks-i$, il y a donc $(n-s-u+1)^{ks-i}$ répartitions différentes.

Il nous reste à calculer la distribution du nombre de transitions atteignant les u sources secondaires de $\mathcal{R}$. Le nombre total des différentes répartitions des transitions partant de S est donné par $\beta(n,u,n-s)$ — voir lemme 29 —.

Comptons le nombre de combinaisons où le nombre de transition dirigées vers les sources de $\mathcal{R}$ est i, on obtient : $\binom{ks}{i}\mathrm{Surj}(i,u)(n-s-u+1)^{ks-i}$.

Donc, partant d'un automate de taille n, ayant s sources et u sources secondaires, la probabilité pour qu'il ait i transitions atteignant ces u sources secondaires est

$$\frac{\binom{ks}{i}\mathrm{Surj}(i,u)(n-s-u+1)^{ks-i}}{\beta(n,u,n-s)}.$$

Théorème 9 *La complexité de l'algorithme est en $O(n^4)$ opérations arithmétiques élémentaires.*

L'algorithme nécessite un précalcule des coefficients $\alpha(n,s)$ et $\beta(n,u,r)$. Une fois ces calculs effectués, tirer un automate nécessite d'exécuter un nombre linéaire d'opérations.

Preuve : L'étape 1) nécessite de connaître la valeur de $\beta(n,u,n-s)$ pour tout $u,s<n$. Pour connaître cette valeur on peut utiliser la formule donné dans le lemme 29 qui est

$$\beta(n,u,n-s)=\sum_{i=0}^{k(n-s)}\binom{n}{i}\mathrm{Surj}(i,u)(n-u-r+1)^{k(n-s)-i}.$$

Le calcul de chaque coefficient demande $O(n)$ opérations et il y a $O(n^3)$ coefficients à calculer. La complexité du calcul des coefficients β est en $O(n^4)$.

Le calcul des coefficients $\alpha(n,s)$ et les valeurs n^s pour $s\leq n$ nécessite un nombre d'opérations élémentaires de $O(n^3)$. Et le calcul des coefficients binomiales demande un nombre quadratique d'opérations. □

Cette preuve n'est pas détaillée et n'explique pas pourquoi seuls les coefficients binomiaux, et les coefficients β et α sont nécessaires à l'algorithme. Nous n'avons pas voulu trop détailler cette preuve car une autre preuve similaire est présentée à la fin de ce chapitre.

Cette complexité est un peu excessive pour un générateur aléatoire uniforme. Les meilleures complexités sont linéaires et les « bonnes » complexités sont quadratiques.

La partie suivante, plus détaillée, présente une autre version du générateur qui possède une complexité théorique en $O(n^3)$. On montre qu'en pratique cette complexité est encore plus faible.

4.2 Algorithme optimisé

Dans cette partie, nous montrons comment réduire la complexité de l'algorithme. Nous commençons par donner une formule permettant de calculer les coefficients $\beta(\cdot,\cdot,\cdot)$ en $O(n^3)$ opérations. Ensuite on discute de la complexité en pratique de l'algorithme qui nous semble inférieure à $O(n^3)$. Enfin nous décrivons en l'algorithme optimisé en pseudo-code.

4.2.1 Calcul de β

Le calcul des valeurs des coefficients $\beta(\cdot,\cdot,\cdot)$ est coûteux. Ce qui suit permet de réduire le coût en calcul des valeurs de β.

Les transitions originaires des sources de l'automate sont toute différentiées et leur nombre est ks si s est le nombre de source de l'automate. La répartition de ces ks transitions vers les états du sous-automate obtenu par l'application η est en bijection avec les applications partant d'un ensemble fini vers un autre ensemble fini et qui vérifient certaines propriétés surjectives.

Définition 23 *Soient trois ensembles finis T, G et U avec $U \subset G$. Les cardinaux des ensembles T, G et U sont respectivement t, g et u avec $0 \leq u \leq g$. On pose $\gamma(t,u,g)$ le nombre d'applications de T vers G qui atteignent tous les éléments de U.*

Avec les notations de la définition ci-dessus, $\gamma(t,u,g)$ est le nombre d'applications f allant de T vers G qui vérifient $U \subset f(T)$.

La remarque précédente sur les transitions partant des sources nous permet de montrer la relation qui suit.

Lemme 32 *Pour des entiers naturels n, u et r on a*

$$\beta(n,u,r) = \gamma\big((n-r)k, u, r+1\big). \tag{4.2}$$

Où pour t,u,g, entiers naturels

$$\begin{aligned}
&\gamma(0,0,g) = 1,\\
&\gamma(t,0,g) = g\cdot\gamma(t-1,0,g),\\
&\gamma(t,u,g) = u\cdot\gamma(t-1,u-1,g) + (g-u)\cdot\gamma(t-1,u,g) \qquad \textit{si } 1 \leq u \leq g, t\\
&\gamma(t,u,g) = 0, \qquad \textit{sinon.}
\end{aligned}$$

Avant de faire la preuve de ce lemme on peut remarquer que le calcul des coefficients $\gamma(t,u,g)$ avec $t,u,g \leq kn$ peut s'effectuer en $O(n^3)$ opérations arithmétiques et que grâce à la relation 4.2 il en est de même pour les coefficients β.

Preuve : Nous avons défini, dans la première section du chapitre, $\beta(n,u,r)$ comme le nombre de répartitions des transitions partant des $n-r$ états sources vers les $r+1$

autres états et qui atteignent toutes les u sources de $\mathcal{R}$. En remarquant qu'il y a $k(n-r)$ transitions et qu'elles sont toutes différentiables, on obtient la relation 4.2.

Appelons Γ l'ensemble des applications considérées. On utilise les notations de la définition 23.

Si T est vide alors l'ensemble des applications de T dans G ne contient que l'application vide. Cette application appartient à Γ que si et seulement si $U = \emptyset$ donc $\gamma(0,0,g) = 1$ et $\gamma(0,u,g) = 0$ si $0 < u$.

Supposons T non vide.

Si U est vide alors toute fonction de T vers G appartient à Γ et donc $\gamma(t,0,g) = g^t$ ce que l'on peut résumer par $\gamma(t,0,g) = g \cdot \gamma(t-1,0,g)$.

Si $t < u$ alors il est impossible que tous les éléments de U soient atteints par une application partant de T donc $|\Gamma| = 0$. Le cas $g < u$ est impossible car $U \subset G$.

On suppose $1 \leq u \leq g, t$. Soit e un élément de T. On partitionne Γ en fonction de l'image de e par une fonction de Γ — ce qui fait en tout g parties —. Soit s un élément de G et Γ_s l'ensemble des fonctions f de Γ telles que $f(e) = s$.

Supposons que s appartienne à U. Nous allons calculer le cardinal de Γ_s, il faut choisir les images des autres éléments de T. Comme s est un élément de U qui est atteint alors il s'agit de choisir les images des éléments de $T \setminus \{e\}$ parmi les éléments de G de telle façon que tous les éléments de $U \setminus \{s\}$ aient au moins un antécédent. Donc dans le cas où $s \in U$ on a $|\Gamma_s| = \gamma(t-1, u-1, g)$.

Supposons que s n'appartienne pas à U. Alors dans ce cas il s'agit de choisir les images des éléments de $T \setminus \{e\}$ parmi l'ensemble G de telle façon que tous les éléments de U aient un antécédent. Donc dans le cas où $s \notin U$ on a $|\Gamma_s| = \gamma(t-1, u, g)$.

Il y a u parties Γ_s qui vérifient $s \in U$, toutes ont $\gamma(t-1, u-1, g)$ comme cardinal. Il y a $g - u$ parties Γ_s qui vérifient $s \notin U$ toutes de cardinal $\gamma(t-1, u, g)$.

Donc $|\Gamma| = u \cdot \gamma(n-1, u-1, g) + (g-u) \cdot \gamma(n-1, u, g)$. □

4.2.2 Stratégie paresseuse

En expérimentant l'algorithme, on se rend compte que les automates acycliques uniformes sont en moyenne plus « longs » que « larges ». On va essayer d'en tirer parti en définissant la largeur d'un automate acyclique.

Définition 24 *La* largeur d'un automate *acyclique est définie récursivement par*

$$\begin{aligned} \operatorname{larg}(\mathcal{A}) &= 0 && \textit{si } \mathcal{A} \textit{ est l'automate vide,} \\ \operatorname{larg}(\mathcal{A}) &= \max\big(\operatorname{src}(\mathcal{A}), \operatorname{larg}(\eta(\mathcal{A}))\big) && \textit{sinon.} \end{aligned}$$

On note $\operatorname{larg}(\mathcal{A})$ *la largeur d'un automate* $\mathcal{A}$.

Une autre façon de définir la largeur est de rassembler les états d'un automate par niveaux et de dire que la largeur est le cardinal du niveau qui contient le plus d'états. Deux états sont dans un même niveau si leurs plus longs chemins qui les

relient à une source sont de même longueur. On voit figure 4.4 l'exemple d'un automate acyclique dont les états ont été rassemblés par niveau. Dans l'algorithme, la largeur de l'automate produit est le maximum des sources tirées à l'étape 2 lors des appels récursifs à l'algorithme.

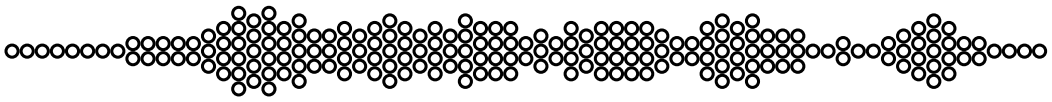


FIGURE 4.4 – Schéma d'un automate acyclique de taille 100 tiré avec l'algorithme. Les cercles représentent les états qui ont été rassemblés par niveau. La largeur de l'automate est 6.

L'expérience montre que la largeur d'un automate est très faible comparé à sa taille n — voir figures 4.5 et 4.4 —.

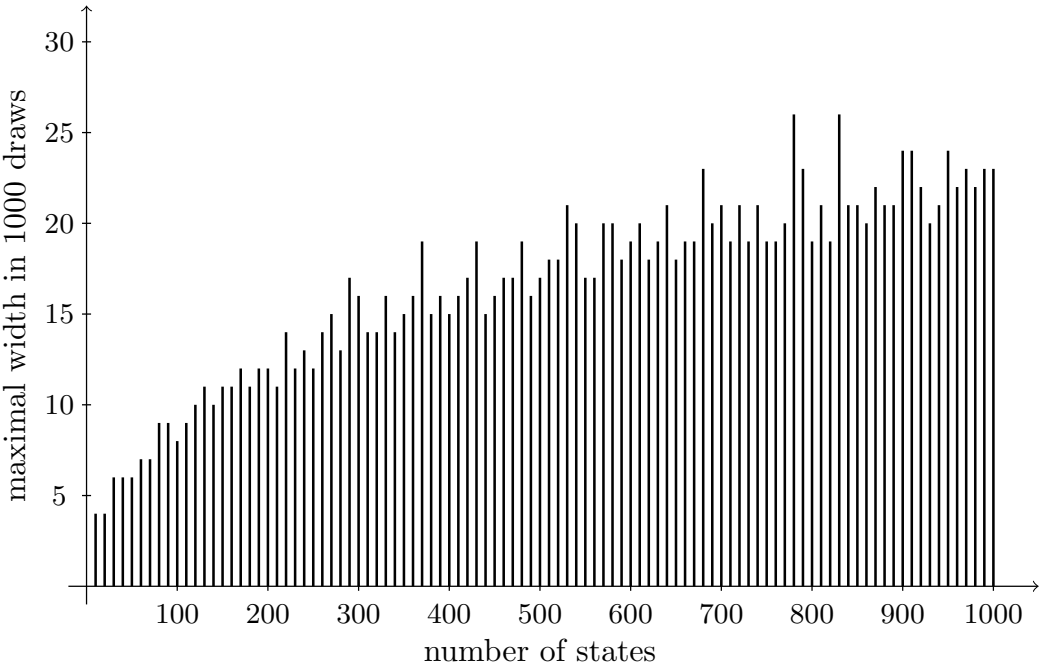


FIGURE 4.5 – Largeur maximal de 1000 automates selon leurs tailles. Les automates ont été tirés uniformément.

On peut tirer profit de cette caractéristique en ne calculant que les coefficients dont on a besoin pour un automate ; le but est de pouvoir montrer le théorème qui suit.

Théorème 10 *La complexité de l'algorithme optimisé est en $O(\omega \cdot n^2)$ opérations arithmétiques élémentaires, où ω est la largeur de l'automate tiré et n sa taille.*

L'algorithme optimisé est une version améliorée de l'algorithme présenté précédemment. Il est décrit en détail à la fin de ce chapitre. Maintenant, nous allons

seulement parler des grandes idées de la preuve. La preuve complète est écrite plus loin, juste après la description de l'algorithme.

La complexité de l'algorithme est liée aux calculs des coefficients binomiaux et des coefficients α et γ. Une fois ces calculs effectués et ces coefficients stockés, tirer un automate de taille n demande un nombre d'opérations en $O(n)$.

L'idée de la « stratégie paresseuse » est de ne calculer une valeur du coefficient γ que lorsqu'on en a besoin au cours de l'exécution. Ainsi si un automate de largeur ω est tiré, comme pour tout u et s on a $u, s \leq \omega$, on montre que l'on aura eu finalement besoin de calculer uniquement $O(\omega^2 \cdot n)$ coefficient γ.

Cependant exécuter l'algorithme nécessite aussi de connaître les valeurs $\alpha(u, s)$; or si on utilise la formule du lemme 30 pour calculer les valeurs $\alpha(u, s)$ on remarque que l'expression de α dépend des coefficients β — donc γ — pour des valeurs de u et s allant jusqu'à n ce qui nécessite de calculer les $O(n^3)$ coefficients γ.

Pour cette raison on utilise une autre expression de α qui est donnée par le lemme suivant.

On rappelle que $\alpha(n, s)$ compte le nombre d'automates acycliques de taille n possédant s sources et $\alpha(n)$ compte le nombre d'automates acycliques de taille n.

Lemme 33 *On a la relation*

$$\alpha(n, s) = \binom{n}{s} \sum_{t=0}^{n-s} (-1)^t \binom{n-s}{t} \alpha(n-s-t) \cdot (n-s-t+1)^{k(s+t)}.$$

La preuve de cette relation est une adaptation d'une preuve de Liskovets dans l'article [Lis06] qui montre que

Théorème 11 *Liskovets*

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

Preuve du théorème 11 : Voir [Lis06] formule (3) où le nombre de puits r est égal à 1. □

Preuve du lemme 33 :

On travaille avec les complétés d'automates acycliques. Soit un sous-ensemble d'état T, $T \subset [n]$ de cardinal t. On s'intéresse aux automates de taille n où tous les éléments de T sont des états sources, on note cet ensemble Δ_T. On précise juste que tous les états de T sont des sources mais il peut exister des sources en qui ne sont pas des états de T. Si on retire les états de T et que l'on normalise les états restant, on conserve un automate acyclique normalisé. L'opération inverse consiste à choisir les directions des transitions partant des états de T qui sont dirigées vers les états restant de l'automate — complété —. On a donc $|\Delta_T| = (n-t+1)^{kt} \alpha(n-t)$.

Soit un ensemble d'état S, $S \subset [n]$ de cardinal s. On pose Φ_S l'ensemble des automates qui ont exactement S comme ensemble d'états sources.

On a $|\Phi_S| = |\Delta_S| - |\bigcup_{S\subsetneq T} \Delta_T|$.

Nous allons calculer le cardinal de ce dernier ensemble. On peut écrire

$$\left|\bigcup_{S\subsetneq T} \Delta_T\right| = \left|\bigcup_{k\in Q\setminus S} \Delta_{S\cup\{k\}}\right|.$$

En utilisant le principe d'inclusion-exclusion on obtient

$$\left|\bigcup_{S\subsetneq T} \Delta_T\right| = \sum_{t=1}^{n-s} (-1)^{t-1} \sum_{\substack{|P|=t \\ P\subset Q\setminus S}} \left|\bigcap_{q\in P} \Delta_{S\cup\{q\}}\right|. \tag{4.3}$$

On remarque que pour tout couple P, Q de sous-ensembles d'états on a $\Delta_P \cap \Delta_Q = \Delta_{P\cup Q}$. En remplaçant correctement cette expression dans la formule 4.3 on obtient

$$\left|\bigcup_{S\subsetneq T} \Delta_T\right| = \sum_{t=1}^{n-s} (-1)^{t-1} \sum_{\substack{|P|=t \\ P\subset Q\setminus S}} |\Delta_{S\cup P}|\,,$$

$$\left|\bigcup_{S\subsetneq T} \Delta_T\right| = \sum_{t=1}^{n-s} (-1)^{t-1} \sum_{\substack{|P|=t \\ P\subset Q\setminus S}} \alpha(n-t-s)(n-t-s+1)^{k(s+t)},$$

$$\left|\bigcup_{S\subsetneq T} \Delta_T\right| = \sum_{t=1}^{n-s} (-1)^{t-1} \binom{n-s}{t} \alpha(n-t-s)(n-t-s+1)^{k(s+t)}.$$

Donc

$$|\Phi_S| = \sum_{t=0}^{n-s} (-1)^{t} \binom{n-s}{t} \alpha(n-t-s)(n-t-s+1)^{k(s+t)}.$$

On remarque — à nouveau — que le cardinal de Φ_S ne dépend que de s et n. Donc en partitionnant les automates acycliques possédant s sources selon les étiquettes de ces sources on obtient $\alpha(n,s) = \binom{n}{s}|\Phi_S|$. □

Lemme 34 *On peut calculer toutes les valeurs de $\alpha(u,s)$ pour $u < n$ et $s < \omega$ en $O(n^2 \cdot \omega)$ opérations arithmétiques.*

Preuve : Selon la formule du lemme 33 le calcul des coefficients $\alpha(n,s)$ nécessite de connaître toutes les valeurs $\binom{i}{j}$ et $\alpha(i)$ pour $i,j \leq n$. Les coefficients binomiaux peuvent être tous calculés à l'avance en $O(n^2)$ opérations avec la relation bien connue $\binom{i}{j} = \binom{i-1}{j} + \binom{i-1}{j-1}$. Les coefficients du type u^s, $u, s < kn$ peuvent être calculés aussi en $O(n^2)$ opérations arithmétiques. Chaque valeur de $\alpha(i)$, $i < n$ peut être calculée en $O(n)$ opérations grâce à la relation donnée dans le théorème 11, donc le calcul complet nécessite $O(n^2)$ opérations arithmétiques. Chaque coefficient $\alpha(i,s)$ est calculé en un nombre linéaire d'opérations grâce à la relation donnée dans le lemme 33. Donc en tout, les coefficients $\alpha(i,s)$ sont calculés en $O(n^2\omega)$ opérations pour $i < n$ et $s < \omega$. □

4.3 Algorithme final

Voici l'algorithme en pseudo-code.

L'algorithme général est `GenerateurAcycliqueSommet(n,s)`, les procédures

— `Recursif`,

— `TirerTransitions`,

— `TirerNombreSourceSecondaires`,

utilisées comme sous-algorithmes, sont décrites à la suite de `GenerateurAcycliqueSommet(n,s)`.

La procédure élémentaire `TirerUniformement` prend en paramètre un ensemble et renvoie un élément choisi aléatoirement de façon uniforme. On rappelle que nous supposons que cette procédure utilise un nombre constant d'opérations lors de son exécution.

La procédure `Ajouter(E,d)` — respectivement `Enlever(E,d)` — renvoie l'ensemble E auquel on a ajouté — respectivement enlevé — l'élément d.

Les automates sont représentés par un ensemble de transitions $p \xrightarrow{\ell} q$ où p et q sont les étiquettes des états et ℓ une lettre de l'alphabet A. Pour un automate $\mathcal{A}$ et une transition $p \xrightarrow{\ell} q$ on note $\mathcal{A} \oplus p \xrightarrow{l} q$ l'automate $\mathcal{A}$ auquel on a ajouté la transition $p \xrightarrow{\ell} q$. L'automate vide est représenté par l'ensemble vide.

Pour tout autres types d'ensemble E on note $|E|$ son cardinal.

Algorithm 3: `GenerateurAcycliqueSommet(n,s)`

input(n *: la taille de l'automate à produire.* s *: le nombre de sources de l'automate.*)
output($\mathcal{A}$ *: un automate acyclique de taille* n.)
local(Q *: un ensemble d'états.* S *: un ensemble d'états renvoyé par la procédure récursive, inutile ici.*)

$Q \longleftarrow [n]$
$\mathcal{A}, S \longleftarrow$ `Recursif`(n, s, Q)
return $\mathcal{A}$

On décrit la procédure `Recursif` qui est la procédure principale. La ligne 6 correspond à l'étape 1 du premier algorithme. C'est l'étape où : ≪ On tire le nombre de sources secondaires ≫. Les lignes 9 à 14 correspondent à l'étape 3 : ≪ On étiquette les sources ≫. La ligne 15 correspond à l'étape 2 : ≪ On tire récursivement un automate acyclique ≫. La ligne 16 correspond à l'étape 4 : ≪ On tire les transitions partant des sources ≫.

Comparé à l'algorithme de la première version, l'ordre des étapes 2 et 3 est interverti. Cela permet de faciliter la description : on choisit d'abord les étiquettes des sommets sources et ainsi on connaît les étiquettes restantes, on peut donc directement tirer un sous-automate possédant les bonnes étiquettes. On évite comme cela de décrire l'étape qui consiste à re-étiqueter le sous-automate. En contrepartie, on doit préciser en paramètre l'ensemble des étiquettes disponibles.

Algorithm 4: `Recursif`(n, s, Q)

input(n *: la taille de l'automate à produire.* s *: le nombre de sources de l'automate.* Q *: un ensemble d'états.*)
output($\mathcal{A}$ *: un automate acyclique de taille* n *à* s *sources.* S *: l'ensemble des états sources de l'automate.*)
local(T *: un ensemble de transitions.* U *: l'ensemble des sources secondaires de* $\mathcal{A}$*.*)
if $n = 0$ **then**
  return $\emptyset$,$\emptyset$
$u \longleftarrow$ `TirerNombreSourceSecondaires`(n, s)
$T \longleftarrow \emptyset$
$S \longleftarrow \emptyset$
for i *allant de* 1 *à* s **do**
  $s \longleftarrow$ `TirerUniformement`(Q)
  $Q \longleftarrow$ `Enlever`(Q, s)
  $S \longleftarrow$ `Ajouter`(S, s)
  for ℓ *dans* A **do**
    $T \longleftarrow$ `Ajouter`$(T, s \xrightarrow{\ell})$
$\mathcal{A}, U \longleftarrow$ `Recursif`$(n - s, u, Q)$
$\mathcal{A} \longleftarrow$ `TirerTransitions`$(\mathcal{A}, T, U, Q)$
return $\mathcal{A}, S$

La procédure `TirerTransitions` prend en paramètre le sous-automate tiré et la liste des transitions à ajouter — sans leurs destinations —, puis renvoie l'automate augmenté de ces transitions que l'on a complétées aléatoirement.

Algorithm 5: `TirerTransitions`$(\mathcal{A}, T, U, Q)$

input($\mathcal{A}$ *: un automate acyclique de taille* $|Q|$*.* T *: un ensemble de transitions non fixées.* U *: les états sources de l'automate* $\mathcal{A}$*.* Q *: les états de l'automate* $\mathcal{A}$*.*)
output($\mathcal{A}$ *: un automate acyclique.*)
local(p *: un entier.* s, p *: des états de* $\mathcal{A}$*.* ℓ *: une lettre.*)
if $T = \emptyset$ **then**
  return $\mathcal{A}$
$s \xrightarrow{\ell} \longleftarrow$ `Choisir`(T)
$p \longleftarrow$ `TirerUniformement`$\Big(\Big[\gamma\big(|T|, |U|, |Q|+1\big)\Big]\Big)$
if $p \leq |U| \cdot \gamma\big(|T|-1, |U|-1, |Q|+1\big)$ **then**
  $q \longleftarrow$ `TirerUniformement`(U)
  $\mathcal{A} \longleftarrow \Big(\mathcal{A} \oplus s \xrightarrow{\ell} q\Big)$
  $\mathcal{A} \longleftarrow$ `TirerTransitions` $\Big(\mathcal{A}, T \setminus \{s \xrightarrow{l}\}, U \setminus \{q\}, Q\Big)$
else
  $q \longleftarrow$ `TirerUniformement`$((Q \setminus U) \cup \{\bot\})$
  $\mathcal{A} \longleftarrow \Big(\mathcal{A} \oplus s \xrightarrow{\ell} q\Big)$
  $\mathcal{A} \longleftarrow$ `TirerTransitions` $\Big(\mathcal{A}, T \setminus \{s \xrightarrow{l}\}, U, Q\Big)$
return $\mathcal{A}$

La procédure récursive `TirerTransitions` utilise la décomposition récursive que nous avons mise en évidence dans la preuve du lemme 32. La procédure reçoit en entrée l'ensemble des transitions à diriger, l'ensemble des états de l'automate et l'ensemble des sources secondaires à couvrir. À chaque appel, la procédure fixe la direction d'une transition, la direction de cette transition est tirée aléatoirement, puis on appelle à nouveau cette procédure sur les transitions restantes. Le choix de la destination de la transition s'effectue soit parmi les sources secondaires, soit parmi les autres états, l'état puits compris. Ces deux cas correspondent à la décomposition évoquée dans la preuve du lemme 32 et dont nous avons calculés les tailles que nous avons rassemblées sous la notation γ. On utilise donc les valeurs des coefficients γ pour pondérer le tirage dans l'un ou l'autre de ces deux cas. Une fois le cas choisi on tire uniformément l'état parmi l'ensemble des états qui correspond au cas choisi — c'est à dire l'ensemble des sources secondaires ou l'ensemble des autres états —.

Voici un récapitulatifs des expressions utiles pour calculer les coefficients nécessaires.

Algorithm 6: `TirerNombreSourceSecondaire`(s, n)

input(s : *le nombre de sources d'un automate.* n : *le nombre d'états d'un automate.*)
output(u : *un nombre aléatoire de sources secondaires d'un automate à* n *états et à* s *sources selon la distribution uniforme des automates acycliques.*)
local(t : *un nombre entier.*)
$t \longleftarrow$ `ChoisirUniformement`$([\alpha(n, s)])$
$u \longleftarrow 0$
while $t > 0$ **do**
  $t \longleftarrow t - \binom{n}{s} \cdot \gamma(ks, u, n - s + 1) \cdot \alpha(n - s, u)$
  $u \longleftarrow u + 1$
return u

Pour les entiers naturels n, s, u et r on a

$$\gamma(0, 0, r) = 1,$$
$$\gamma(s, 0, r) = r \cdot \gamma(s - 1, 0, r) \qquad \text{si } s > 0,$$
$$\gamma(s, u, r) = u \cdot \gamma(s - 1, u - 1, r) + (r - u) \cdot \gamma(s - 1, u, r) \qquad \text{si } 1 \leq u \leq r, s,$$
$$\gamma(s, u, r) = 0 \qquad \text{sinon,}$$

$$\alpha(n, s) = \binom{n}{s} \sum_{t=0}^{n-s} (-1)^t \binom{n-s}{t} \alpha(n - s - t) \cdot (n - s - t + 1)^{k(s+t)},$$

$$\alpha(n) = \sum_{t=0}^{n-1} (-1)^{n-t-1} \binom{n}{t} (t + 1)^{k(n-t)} \alpha(t),$$

$$\binom{n}{s} = \binom{n-1}{s-1} + \binom{n-1}{s} \qquad \text{si } 1 \leq s < n,$$

$$\binom{n}{0} = 1,$$

$$\binom{n}{n} = 1 \qquad \text{si } s = n,$$

$$\binom{n}{s} = 0 \qquad \text{sinon ,}$$

$$n^s = \gamma(s, 0, n).$$

Maintenant que l'algorithme est décrit on calcule sa complexité.

Preuve du théorème 10 : Les coefficients utilisés dans l'algorithme sont γ et α. Nous avons montré que le nombre de calculs nécessaires pour obtenir les valeurs $\alpha(n, s)$, avec $s \leq \omega$, est en $O(\omega \cdot n^2)$. — voir lemme 34 page 90 —. Le calcul de chaque coefficient $\gamma(t, u, g)$ avec $t, u \leq k\omega$ nécessite un nombre constant d'opérations donc les valeurs γ peuvent être calculées avec en tout $O(n \cdot \omega^2)$ opérations.

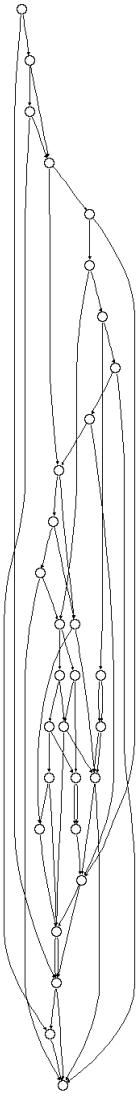
Une fois que les valeurs des coefficients α et γ sont connues, l'algorithme tire le nombre de sources secondaires avec la procédure `TirerNombreSourceSecondaire`. La boucle de cette procédure prend moins de ω opérations et en faisant la somme des passages dans cette boucle au cours des différents appels récursifs, on retrouve le nombre d'états n. Ensuite l'algorithme tire uniformément les étiquettes des s sources, dans s itérations de boucle, donc n fois au cours des différents appels récursifs. L'algorithme fait ensuite appel à la procédure `TirerTransitions`, qui est appelée récursivement autant de fois que le cardinal de l'ensemble T son paramètre. $|T|$ est le nombre de transitions partant des sources de l'automate. Au cours de tous les appels récursifs la procédure est utilisée autant de fois qu'il y a de transitions dans l'automate c'est à dire kn fois. À chaque appel récursif de l'algorithme, le paramètre n — la taille — est décrémenté au moins de 1, le nombre d'appels est donc inférieur ou égal à n. L'exécution propre de l'algorithme, sans compter les calculs des coefficients, nécessite $O(n)$ opérations. Donc la complexité totale de l'algorithme est en $O(n + \omega^2 n + \omega n^2) = O(\omega n^2)$ opérations. □

L'algorithme que nous venons de décrire produit un automate acyclique déterministe tiré uniformément sans d'autre contrainte que sa taille. On peut, cependant, sans trop modifier l'algorithme, ajouter des contraintes comme l'accessibilité ou en modifiant le nombre de puits — qui est fixé à 1 dans l'algorithme —.

Pour rendre l'automate accessible, il suffit, de fixer à 1 le nombre de sources de l'automate en utilisant directement la procédure `Recursif` avec les paramètres n, 1 et $[n]$.

Cette modification, grâce à la remarque faite au lemme 1, permet de produire un générateur d'automates déterministes acycliques accessibles non étiquetés.

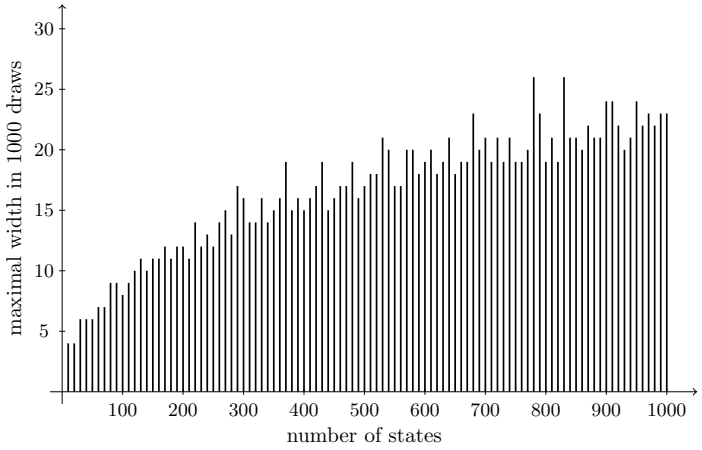
Conclusion et perspectives sur les automates acycliques



Dans les chapitres 4 et 3 nous avons vus deux procédés pour générer des automates acycliques sur un alphabet fixé — à gauche un exemple d'automate de taille 30 généré avec l'algorithme —. Même si elle est plus élégante et plus souple, le générateur par chaîne de Markov s'avère moins utilisable en pratique à cause de ce problème du temps de mélange.
Nous avons également vu que la version paresseuse de l'algorithme récursif fonctionnait particulièrement bien si les automates générés ont une faible largeur. Même si nous n'avons pas encore pu le prouver, il semble que ce soit le cas. Voici un exemple de la répartition des différents niveaux d'un automate acyclique aléatoire dans la figure ci-dessous : l'état initial est à gauche et on a représenté sur une même colonne les états qui sont à la même distance de l'état le plus à gauche.



On voit que l'automate possède une faible largeur ce qui signifie qu'on ne calcul les coefficients β que pour des valeurs faibles de s et qu'on est significativement en dessous de la complexité cubique. Cela est également visible sur l'expérimentation ci dessous où nous avons calculé la largeur maximum sur un échantillon de 1000 automates pour différentes tailles. On observe une croissance plus logarithmique que linéaire.



Nous avons utilisé le générateur aléatoire pour tester la proportion d'automates minimaux parmi les acycliques. On peut voir les résultats sur la figure 4.3.

Elle semble montrer que parmi les automates acycliques la proportion de minimaux est constante. Cette propriété si elle est vérifiée nous suggère un algorithme de génération aléatoire d'automates minimaux par rejet. Le principe est de tirer des automates acycliques jusqu'à en obtenir un qui soit minimal.

Nous avons également regardé combien, en moyenne, il y avait de mots reconnus par un automate acyclique aléatoire

Les deux algorithmes ont été implanté en Python (150 lignes de code).

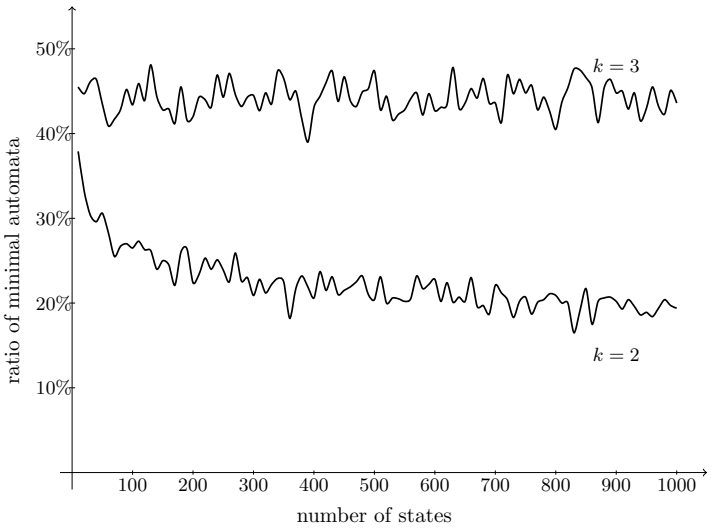


FIGURE 4.6 – Proportion d'automates minimaux parmi les automates générés en fonction de la taille sur un échantillon de 1000 automates répartis selon des tailles allant de 10 à 100.

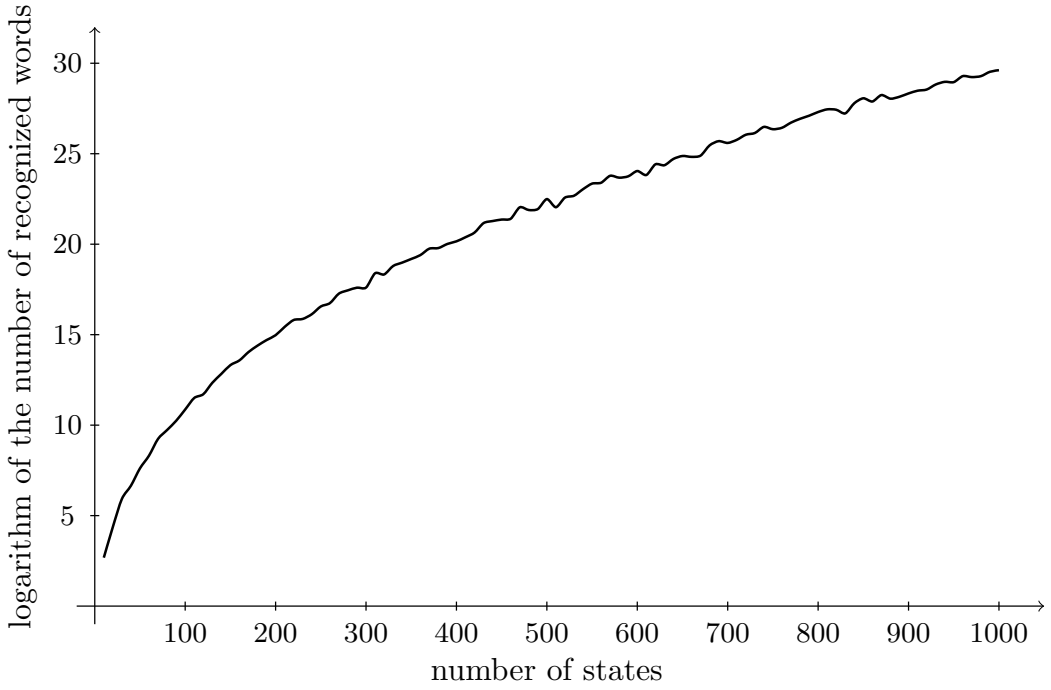


FIGURE 4.7 – Logarithme naturel du nombre de mots reconnus par des automates générés uniformément. L'alphabet est de cardinal 2 et le nombre de mots est fonction de la taille de l'automate. Le nombre d'automates générés est 1000.

Index

β-uniforme, 32

a-cycle, 17
accessible, 18
acyclique, 17
algorithme de déterminisation, 22
alphabet, 14
ancêtre, 67
automate, 15
automate accessible, 18
automates étiquetés, 20
automate minimal, 22
automate vide, 76
automates non étiquetés, 20

chaîne apériodique, 55
chaîne de Markov, 52
chaine symétrique, 55
chemin, 16
classes d'isomorphismes, 20
co-accessible, 18
combinatoire, 12
complété, 18
complexité dans le pire des cas, 12
complexité en moyenne, 13
complexité générique, 13
cycle, 10
cycle accessible, 16
cycle d'un automate, 16
cycle primitif, 17

dénombrable, 9
déterminisation, 21
déterminisé, 22
déterministe complet, 18
déterministe, 18
directement équivalents, 66
distribution, 10
distribution initiale, 52
distribution stationnaire, 55
distribution uniforme, 11
domine, 10

écart type, 11
émondé, 18
émonder, 18
en moyenne, 13
ensemble correspondant, 19
ergodique, 55
état initial, 15
état puits, 18
état terminal, 15
états équivalents, 22
états, 15
évènements, 11

facteur primitif, 15
fonction de transitions d'un automate, 18
fusionner, 23

générateur aléatoire, 14
générique, 12
grand cycle, 33
graphe associé, 15

irréductible, 54
isomorphe, 20

langage, 15
langage miroir, 17
langage reconnu, 16
langage reconnu par un état, 22
langages rationnels, 15
largeur d'un automate, 87
lettres, 14
longueur d'un chemin, 16
longueur d'un mot, 15

mot, 14
mot associé à un chemin, 16
mot miroir, 17
mot primitif, 15
mot reconnu, 16

négligeable, 12

opération arithmétique, 13
opérations élémentaires, 12
orbite, 10

partie accessible, 18
période d'un cycle, 17
permutations, 10
place apériodique, 54
places, 52
probabilité conditionnelle, 11
probabilités, 11
probabilités de transitions, 52
puissance d'un mot, 15

rang d'un état, 66

source, 16
source secondaire, 77
sous-automate, 19
sous-automate normalisé, 19
structure de transition, 15, 31
super-polynomial, 10

taille d'un automate, 15
taille d'un objet, 12
taille d'une application, 10
temps d'exécution, 12
temps de mélange, 55
trace, 35
transitions, 15

variable aléatoire, 11
variance, 11

Bibliographie

[AMR08] Marco Almeida, Nelma Moreira, and Rogério Reis. Exact generation of minimal acyclic deterministic finite automata. *Int. J. Found. Comput. Sci.*, 19(4) :751–765, 2008. (Cité à la page 49.)

[BBCF10] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata. *arXiv :1010.5318*, abs/1010.5318, 2010. (Cité à la page 25.)

[BDN09] Frédérique Bassino, Julien David, and Cyril Nicaud. On the average complexity of moore's state minimization algorithm. In *STACS*, pages 123–134, 2009. (Cités aux pages 6, 27 et 46.)

[BDN12] Frédérique Bassino, Julien David, and Cyril Nicaud. Average case analysis of moore's state minimization algorithm. *Algorithmica*, 63(1-2) :509–531, 2012. (Cités aux pages 6, 27 et 46.)

[BDS12] Frédérique Bassino, Julien David, and Andrea Sportiello. Asymptotic enumeration of minimal automata. In *STACS*, pages 88–99, 2012. (Cité à la page 6.)

[BN07] Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. *Theor. Comput. Sci.*, 381(1-3) :86–104, 2007. (Cité à la page 6.)

[Brz62] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*, pages 529–561. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962. Volume 12 of MRI Symposia Series. (Cités aux pages 6, 23, 26 et 28.)

[CCM09] Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Small extended expressions for acyclic automata. In Sebastian Maneth, editor, *CIAA*, volume 5642 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2009. (Cité à la page 49.)

[CCM10] Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Acyclic automata and small expressions using multi-tilde-bar operators. *Theor. Comput. Sci.*, 411(38-39) :3423–3435, 2010. (Cité à la page 49.)

[CF11] Vincent Carnino and Sven De Felice. Random generation of deterministic acyclic automata using markov chains. In *CIAA*, pages 65–75, 2011. (Cités aux pages 7 et 51.)

[CF12] Vincent Carnino and Sven De Felice. Sampling different kinds of acyclic automata using markov chains. *Theor. Comput. Sci.*, 450 :31–42, 2012. (Cités aux pages 7 et 51.)

[CN12] Arnaud Carayol and Cyril Nicaud. Distribution of the number of accessible states in a random deterministic automaton. In *STACS*, pages 194–205, 2012. (Cités aux pages 6, 37 et 47.)

[CP05] Jean-Marc Champarnaud and Thomas Paranthoën. Random generation of DFAs. *Theor. Comput. Sci.*, 330(2) :221–235, 2005. (Cité à la page 6.)

[Dav10] Julien David. The average complexity of moore's state minimization algorithm is $o(n \log \log n)$. In *MFCS*, pages 318–329, 2010. (Cités aux pages 6, 27 et 46.)

[Dav12] Julien David. Average complexity of moore's and hopcroft's algorithms. *Theor. Comput. Sci.*, 417 :50–65, 2012. (Cités aux pages 6, 27 et 46.)

[DFLS04] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5) :577–625, 2004. (Cité à la page 14.)

[DKS01] Michael Domaratzki, Derek Kisman, and Jeffrey Shallit. On the number of distinct languages accepted by finite automata with n states. In *DCFS*, pages 67–78, 2001. (Cité à la page 49.)

[Dom02] Michael Domaratzki. Improved bounds on the number of automata accepting finite languages. In *Developments in Language Theory*, pages 209–219, 2002. (Cité à la page 49.)

[ER59] P. Erdös and A. Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6 :290–297, 1959. (Cité à la page 47.)

[ET67] Paul Erdos and Paul Turán. On some problems of a statistical group theory, III. *Acta Math. Acad. Sci. Hungar*, 18(3-4) :309–320, 1967. (Cités aux pages 34 et 44.)

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, January 1968. (Cités aux pages 10 et 52.)

[FN13a] Sven De Felice and Cyril Nicaud. Brzozowski algorithm is generically super-polynomial for deterministic automata. In *Developments in Language Theory*, pages 179–190, 2013. (Cités aux pages 6 et 25.)

[FN13b] Sven De Felice and Cyril Nicaud. Random generation of deterministic acyclic automata using the recursive method. In *CSR*, pages 88–99, 2013. (Cités aux pages 7 et 75.)

[FO89] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1989. (Cité à la page 40.)

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. (Cités aux pages 5, 12, 40 et 42.)

[FZC94] Philippe Flajolet, Paul Zimmermann, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theor. Comput. Sci.*, 132(2) :1–35, 1994. (Cités aux pages 14 et 75.)

[Goo61] Irving John Good. An asymptotic formula for the differences of the powers at zero. *Ann. Math. Statist.*, 32 :249–256, 1961. (Cité à la page 38.)

[Hoa62] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1) :10–16, January 1962. (Cités aux pages 5 et 26.)

[Hop71] John E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The Theory of Machines and Computations*, pages 189–196. Academic Press, 1971. (Cités aux pages 6, 23 et 26.)

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. (Cité à la page 14.)

[Knu73a] Donald E. Knuth. *The Art of Computer Programming, Volume I : Fundamental Algorithms, 2nd Edition*. Addison-Wesley, 1973. (Cité à la page 13.)

[Knu73b] Donald E. Knuth. *The Art of Computer Programming, Volume III : Sorting and Searching*. Addison-Wesley, 1973. (Cité à la page 13.)

[Knu81] Donald E. Knuth. *The Art of Computer Programming, Volume II : Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981. (Cités aux pages 13 et 14.)

[Kor78] Aleksey Korshunov. Enumeration of finite automata. *Problemy Kibernetiki*, 34 :5–82, 1978. in russian. (Cités aux pages 6 et 37.)

[Lis06] Valery A. Liskovets. Exact enumeration of acyclic deterministic automata. *Discrete Applied Mathematics*, 154(3) :537–551, 2006. (Cités aux pages 49 et 89.)

[LPW06] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006. (Cités aux pages 52 et 55.)

[MDBM01] Guy Melançon, Isabelle Dutour, and Mireille Bousquet-Mélou. Random generation of directed acyclic graphs. *Electronic Notes in Discrete Mathematics*, 10 :202–207, 2001. (Cité à la page 51.)

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956. (Cités aux pages 23 et 25.)

[Nic99] Cyril Nicaud. Average state complexity of operations on unary automata. In *MFCS*, pages 231–240, 1999. (Cité à la page 6.)

[NW75] A. Nijenhuis and H.S. Wilf. *Combinatorial algorithms*. Computer science and applied mathematics. Academic Press, 1975. (Cités aux pages 14 et 75.)

[PW96] James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Struct. Algorithms*, 9(1-2) :223–252, 1996. (Cité à la page 73.)

[Rev92] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theor. Comput. Sci.*, 92(1) :181–189, 1992. (Cités aux pages 49, 66 et 68.)

[Ré85] Jean-Luc Rémy. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 19(2) :179–195, 1985. (Cité à la page 14.)

[Sta00] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge Univ. Pr., 2000. (Cité à la page 49.)

[TV05] Deian Tabakov and Moshe Y. Vardi. Experimental evaluation of classical automata constructions. In *Proceedings of the 12th international conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'05, pages 396–411, Berlin, Heidelberg, 2005. Springer-Verlag. (Cité à la page 47.)

Automates codéterministes et automates acycliques : analyse d'algorithme et génération aléatoire

---

Résumé :

Le cadre générale de cette thèse est l'analyse quantitative des objets issus de la théorie des langages rationnels. On adapte des techniques d'analyse d'algorithmes — complexité en moyenne, complexité générique, génération aléatoire,... — à des objets et à des algorithmes qui font intervenir des classes particulières d'automates.

Dans une première partie nous étudions la complexité de l'algorithme de minimisation de Brzozowski. Bien qu'ayant une mauvaise complexité dans le pire des cas, cet algorithme a la réputation d'être efficace en pratique. En utilisant les propriétés typiques des applications et des permutations aléatoires, nous montrons que la complexité générique de l'algorithme de Brzozowski appliqué à un automate déterministe croît plus vite que tout polynôme en n, où n est le nombre d'états de l'automate.

Dans une seconde partie nous nous intéressons à la génération aléatoire d'automates acycliques. Ces automates sont ceux qui reconnaissent les ensembles finis de mots et sont de ce fait utilisés dans de nombreuses applications, notamment en traitement automatique des langues. Nous proposons deux générateurs aléatoires. Le premier utilise le modèle des chaînes de Markov, et le second utilise la « méthode récursive », qui tire partie des décompositions combinatoires des objets pour faire de la génération. La première méthode est souple mais difficile à calibrer, la seconde s'avère relativement efficace. Une fois implantée, cette dernière nous a notamment permis d'observer les propriétés typiques des grands automates acycliques aléatoires.

Mots clés : Algorithme de Brzozowski, Automates acycliques, Structures combinatoire, Probabilités discrètes, Génération aléatoire.

Codeterministic automata and acyclic automata : analysis of algorithm and random generation

Abstract :

The general context of this thesis is the quantitative analysis of objects coming from rational language theory. We adapt techniques from the field of analysis of algorithms (average-case complexity, generic complexity, random generation...) to objects and algorithms that involve particular classes of automata.

In a first part we study the complexity of Brzozowski's minimisation algorithm. Although the worst-case complexity of this algorithm is bad, it is known to be efficient in practice. Using typical properties of random mappings and random permutations, we show that the generic complexity of Brzozowski's algorithm grows faster than any polynomial in n, where n is the number of states of the automaton.

In a second part, we study the random generation of acyclic automata. These automata recognize the finite sets of words, and for this reason they are widely use in applications, especially in natural language processing. We present two random generators, one using a model of Markov chain, the other a "recursive method", based on a cominatorics decomposition of structures. The first method can be applied in many situations but is very difficult to calibrate, the second method is more efficient. Once implemented, this second method allows to observe typical properties of acyclic automata of large size.

Keywords : Brzozowski's algorithm, Acyclic automata, Combinatorial structures, Discretes probability, Random generation