# Algorithmes, mots et textes aléatoires

Julien Clément

# ALGORITHMES, MOTS ET TEXTES ALÉATOIRES

JULIEN CLÉMENT

Mémoire d'habilitation à diriger des recherches

présenté le **12 décembre 2011**

**Rapporteurs**

Mireille Bousquet-Mélou, Directrice de recherche, CNRS, Labri (Bordeaux)
Luc Devroye, Professeur, McGill University (Montréal)
Philippe Jacquet, Directeur de recherche (INRIA Rocquencourt)

**Examinateurs :**

Marie-Pierre Béal, Professeur, LIGM (Paris-Est)
Mireille Bousquet-Mélou, Directrice de recherche, CNRS, Labri (Bordeaux)
Luc Devroye, Professeur, McGill University (Montréal)
Philippe Jacquet, Directeur de recherche (INRIA Rocquencourt)
Conrado Martinez, Professeur, UPC (Barcelone)
Brigitte Vallée, Directrice de recherche, CNRS, Greyc (Caen)

UFR de Sciences
Université de Caen Basse-Normandie

Julien Clément : *Algorithmes, mots et textes aléatoires*, Mémoire d'habilitation à diriger des recherches, © 12 décembre 2011

Ce mémoire est mis en forme avec le style `classicthesis`.

# REMERCIEMENTS

J'ai un peu hésité à écrire ces remerciements, l'usage en vigueur pour les habilitations permettant de se passer de cet exercice délicat...

Cependant cette soutenance se caractérise pour moi par une absence : celle de Philippe Flajolet. En effet, aujourd'hui encore le fait que ma soutenance d'habilitation puisse se faire sans lui me paraît irréel. Avec le recul je réalise véritablement la chance que j'ai eue de le côtoyer.

Je suis par exemple irrémédiablement marqué par le voyage effectué en 1998 à l'occasion de la conférence SODA à San Francisco, où encore jeune thésard et naïf, Philippe m'avait littéralement pris sous son aile. Il m'a emmené rencontrer Don Knuth à Stanford, voir les séquoias géants ou encore m'a expliqué les mœurs américaines (nous sommes en pleine affaire Monica Lewinski). J'ai été très heureux de le retrouver treize ans plus tard en janvier 2011 dans des circonstances quasi-identiques (même conférence dans le même hôtel et la même ville), et ce, toujours avec le même plaisir, aussi bien pour aller boire une bière, manger un filet de bœuf, discuter d'un vieil épisode de l'inspecteur Columbo ou bien sûr discuter de science. J'ai évidemment une pensée pour lui comme plusieurs d'entre-vous j'imagine.

Je remercie aussi bien sûr Brigitte Vallée pour son soutien sans faille à la fois depuis le début de ma «carrière» et durant cette dernière année qui a été très particulière. Je suis bien incapable de faire montre de son énergie et de son implication dans tout ce qu'elle entreprend. Cela me laisse toujours aussi pantois après toutes ces années !

Mes remerciements vont également à Mireille-Bousquet Mélou, Luc Devroye et Philippe Jacquet pour avoir accepté d'être rapporteurs, et à Marie-Pierre Béal, Conrado Martinez (et Brigitte Vallée mais cela va de soit) d'être examinateurs. J'ai à la fois beaucoup d'estime scientifique mais aussi de l'affection pour chacun d'eux. Il correspondent à l'idée que je me fais de la recherche scientifique dans une atmosphère ouverte.

Enfin, désolé pour tous ceux qui resteront ici anonymes, mais il serait trop périlleux de remercier sans en oublier toutes les personnes (personnels administratifs, collègues, amis, et bien sûr famille [1]) qui ont contribué à rendre cette dernière décennie si intéressante. J'espère qu'ils se reconnaîtront tous individuellement...

À tous, merci !

1. Encore qu'il me paraît délicat de ne pas citer Gwenaëlle qui me supporte au quotidien et Jérémiah qui vient régulièrement égayer le tableau blanc de mon bureau.

# AVANT-PROPOS

L'objet principal de ce mémoire est le *mot*, le plus souvent *aléatoire*.

J'ai choisi de découper ce document en cinq chapitres, dont, mis à part le premier chapitre introductif aux différents domaines abordés, les quatre autres chapitres recouvrent différents aspects de mes travaux depuis 10 ans. Ces chapitres traduisent en fait une évolution thématique et géographique. Après une thèse au GREYC à Caen qui concernait l'analyse des tries (aussi parfois nommés arbres digitaux) avec des sources dynamiques, j'ai obtenu un poste en 2001 au laboratoire Gaspard Monge à Marne-la-Vallée, où j'ai pu profiter des compétences locales en théorie des langages et en théorie de l'information. Revenu à Caen en 2006, mais favorablement impressionné par ce que j'avais appris à Marne, j'essaie depuis de mélanger les points de vue analytique (plutôt continu) et combinatoire (plus discret).

Le premier chapitre commence par une présentation générale de l'analyse en moyenne d'algorithmes et la combinatoire analytique, puisque l'essentiel de mes résultats se range dans cette catégorie. Ensuite, la théorie des langages et des automates intervient forcément : d'abord parce que c'est une approche fondamentale pour l'étude des mots vus comme des suites de lettres sur un alphabet ; ensuite par la similitude formelle entre les expressions régulières et les classes combinatoires considérées en combinatoire analytique. Enfin, si la théorie des langages cerne bien la combinatoire des mots, la théorie de l'information fournit un autre point de vue sur la modélisation des mots avec la notion de source.

Le deuxième chapitre s'intéresse plutôt à l'aspect combinatoire des mots, tout en gardant en arrière plan l'idée d'explorer des modèles probabilistes pour les mots un peu différents du modèle uniforme (et le plus utilisé) : les mots de Lyndon sont vus comme des mots circulaires ou «colliers» ; les tables de préfixes décrivent les mots en ne regardant que les propriétés de recouvrement d'un mot avec lui-même ; enfin l'étude d'une famille de code préfixes infinis explore un autre aspect en construisant un ensemble de mots (ou code) obéissant à plusieurs contraintes (condition préfixe et optimalité par rapport à la longueur moyenne du code).

Dans le troisième chapitre on s'intéresse aux statistiques d'occurrence d'un motif dans un texte aléatoire. Ici le motif est assez général, dans le sens où on peut considérer n'importe quel ensemble fini de mots finis, alors que le cadre habituel se restreint aux ensemble de mots *réduits* (où aucun mot n'a le droit d'être facteur d'un autre). L'accent est mis sur l'utilisation de la méthode d'inclusion-exclusion qui fournit une approche particulièrement élégante pour ce problème.

Le dernier chapitre «remet en cause» un paradigme pourtant central en analyse d'algorithmes. Est-il légitime de considérer des données «atomiques» (dans le sens informatique du terme, i.e., indivisibles) dans le modèle de calcul ? Le calcul ne devrait-il pas être vu

à son niveau le plus fondamental, c'est-à-dire, par la manipulation de symboles élémentaires tels que le bit, l'octet, le caractère ou le mot-machine ? Par exemple, les structures de trie et d'arbre binaires de recherche ne considèrent pas le même type de données : la première travaille sur des chaînes de caractères tandis que la deuxième considère plutôt un univers de clefs où le coût d'une comparaison est unitaire. L'idée de ce chapitre est de considérer que toute donnée est produite par une source sous la forme d'une suite de symboles, et d'analyser des algorithmes classiques pour évaluer l'impact de cette représentation sur les performances en ne considérant que des comparaisons, élémentaires, de symboles.

# LISTE DE PUBLICATIONS

Les articles suivant sont téléchargeables au format PDF à l'URL «http://users.info.unicaen.fr/~clement/publications/».

*Revues internationales avec comité de lecture*


[1] Bassino, F., Clément, J., and Nicaud, C. The standard factorization of Lyndon words : an average point of view. *Discrete Mathematics 290*, 1 (2005), 1–25.

[2] Bassino, F., Clément, J., and Nicodème, P. Counting occurrences for a finite set of words : combinatorial methods. *Transactions in Algorithms* (2010). To appear (accepted for publication).

[3] Boeva, V., Clément, J., Régnier, M., Roytberg, M., and Makeev, V. Exact p-value calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cis-regulatory modules. *Algorithms for molecular biology 2*, 13 (2007), 25 pages.

[4] Cesaratto, E., Clément, J., Daireaux, B., Lhote, L., Maume-Deschamps, V., and Vallée, B. Regularity of the euclid algorithm ; application to the analysis of fast gcd algorithms. *J. Symb. Comput. 44*, 7 (2009), 726–767.

[5] Clément, J., Duval, J.-P., Guaiana, G., Perrin, D., and Rindone, G. Parsing with a finite dictionary. *Theoretical Computer Science 340*, 1 (2005), 432–442.

[6] Clément, J., Flajolet, P., and Vallée, B. Dynamical sources in information theory : a general analysis of tries structures. *Algorithmica 29* (2001), 307–369. (special issue).


*Actes de colloques internationaux avec comité de lecture*


[7] Archibald, M., and Clément, J. Average depth in a binary search tree with repeated keys. In *Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities* (Nancy, 2006). (11 pages).

[8] Bassino, F., Clément, J., Fayolle, J., and Nicodème, P. Counting occurrences for a finite set of words : an inclusion-exclusion approach. In *Proceedings of the 2007 Conference on Analysis of Algorithms* (2007), P. Jacquet, Ed., DMTCS, proc. AH, pp. 29–44. Proceedings of a colloquium organized at Juan-les-Pins, France, June 2007.

[9] Bassino, F., Clément, J., Fayolle, J., and Nicodème, P. Constructions for clumps statistics. In *Fifth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities* (Blaubeuren, 2008).

[10] Bassino, F., Clément, J., and Nicaud, C. The average lengths of the factors of the standard factorization of Lyndon words. In *Developments in Language Theory* (2003), M. Ito and M. Toyama, Eds., vol. 2450 of

*Lecture Notes in Comput. Sci.*, Springer, pp. 307–318. 6th International Conference, DLT, Kyoto, Japan, September 18-21, 2002.

[11] Bassino, F., Clément, J., and Nicaud, C. Lyndon words with a fixed standard right factor. In *SODA* (2004), J. I. Munro, Ed., SIAM, pp. 646–647. Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004.

[12] Bassino, F., Clément, J., Seroussi, G., and Viola, A. Optimal prefix codes for pairs of geometrically-distributed random variables. In *2006 IEEE International Symposium on Information Theory* (Seattle, WA, 2006). 5 pages.

[13] Bassino, F., Clément, J., Seroussi, G., and Viola, A. Optimal prefix codes for some families of two-dimensional geometric distributions. In *2006 Data Compression Conference (DCC 2006), 28-30 March 2006, Snowbird, UT, USA* (2006), IEEE Computer Society, pp. 113–122.

[14] Boeva, V., Clément, J., Régnier, M., and Vandenbogaert, M. Assessing the significance of sets of words. In *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings* (2005), vol. 3537 of *Lecture Notes in Computer Science*, Springer, pp. 358–370.

[15] Clément, J., Crochemore, M., and Rindone, G. Reverse Engineering Prefix Tables. In *Symposium on Theoretical Aspects of Computer Science (STACS)* (2009), pp. 289–300.

[16] Clément, J., Flajolet, P., and Vallée, B. The analysis of hybrid trie structures. In *Proceedings of the Ninth Annual ACM–SIAM Symposium on Discrete Algorithms* (Philadelphia, 1998), SIAM Press, pp. 531–539.

[17] Clément, J., Lhote, L., and Vallée, B. Entropy for dynamical sources. In *Proceedings of IWAP 2008* (Compiègne, France, 2008). 10 pages, Proceedings sur CD.

[18] Vallée, B., Clément, J., Fill, J. A., and Flajolet, P. The number of symbol comparisons in quicksort and quickselect. In *ICALP 2009* (2009), S. A. et al. (Eds.), Ed., vol. Part I, LNCS 5555, pp. 750–763.

*Autres*

[19] Bassino, F., Clément, J., Seroussi, G., and Viola, A. Optimal prefix codes for pairs of geometrically-distributed random variables. 38 pages, soumis à IEEE Transactions In Information Theory.

[20] Clément, J. *Arbres digitaux et sources dynamiques*. PhD thesis, Université de Caen, 2000.

# TABLE DES MATIÈRES

# 1 CONTEXTE

Dans ce chapitre, je présente succinctement le domaine de l'analyse en moyenne et de la combinatoire analytique, la théorie des langages et, en théorie de l'information, la notion de source de symboles. Le cadre et les notions introduites seront utilisées dans la suite du mémoire.

## 1.1 ANALYSE D'ALGORITHMES

L'analyse d'un algorithme consiste à déterminer les ressources de calcul nécessaires (le plus souvent temps et espace) à l'exécution de l'algorithme. La taille d'une donnée est en général reliée assez « naturellement » à la place mémoire utilisée. Ainsi la taille d'un tableau sera souvent le nombre d'éléments du tableau. Pour un entier $m$, on aura recours au nombre de bits nécessaires à sa représentation. La taille d'un mot sera sa longueur, et celle d'un arbre sera le nombre de nœuds, etc. Une fois la taille définie, on regroupe les données d'un ensemble $\mathcal{C}$ par taille, et l'on considère l'ensemble des données de taille $n$

$$\mathcal{C}_n = \{x \in \mathcal{C} \ ; \ |x| = n\},$$

où $|x|$ désigne la taille de $x$.

On associe alors à un algorithme $\mathcal{A}$ fonctionnant sur cet ensemble de données un paramètre de coût $c$ défini sur $\mathcal{C}$ (en général à valeurs entières) relatif soit à l'exécution même de cet algorithme sur la donnée $x$ (place mémoire, nombre d'opérations fondamentales comme des comparaisons, des affectations...), soit à la configuration de sortie produite (comme le degré du PGCD dans le calcul du PGCD de 2 polynômes de $\mathbb{Z}[X]$ par l'algorithme d'Euclide). Une analyse de complexité cherche à caractériser l'évolution d'un paramètre en fonction de la taille $n$ de la donnée. Si $c_n$ est la restriction de $c$ à $\mathcal{C}_n$, on peut définir la complexité dans le meilleur des cas et dans le pire des cas, $M_n$ et $P_n$ par

$$M_n = \inf\{c_n(\gamma) \mid \gamma \in C_n\}, \quad P_n = \sup\{c_n(\gamma) \mid \gamma \in C_n\}.$$

Ces deux notions sont intéressantes puisqu'elles donnent un encadrement du paramètre $c_n$. L'analyse de complexité en moyenne, c'est-à-dire l'étude de $c_n$ sur $\mathcal{C}_n$ comme *variable aléatoire*, permet de préciser encore les choses. Il faut définir bien sûr un modèle probabiliste sur $\mathcal{C}_n$ spécifiant la répartition des données en entrée. La variable $c_n$ devient une variable aléatoire. Par exemple l'espérance de la variable $c_n$ :

$$\mathbf{E}[c_n] = \sum_k k P[c_n = k],$$

donne une information sur le comportement de l'algorithme *en moyenne* sur des données de taille $n$. On peut bien sûr aller plus loin et analyser également les autres moments ou encore la distribution.

Le modèle probabiliste le plus utilisé et le plus simple pour les entrées est le modèle uniforme sur $\mathcal{C}_n$. Un exemple classique est celui des tris à base de comparaisons qui considère $n$ nombres réels tirés de façon indépendante selon une loi de probabilité continue sur $[0, 1]$. En raisonnant directement sur l'ordre, cela revient [1] à choisir une permutation de manière uniforme dans $\{1, \dots, n\}$.

Les diverses notions d'analyse (pire des cas, meilleur des cas et cas en moyenne) sont bien distinctes. Un algorithme très peu efficace sur certaines configurations dans le pire des cas et pratiquement linéaire en moyenne, constitue en fait un algorithme efficace la plupart du temps (donc utilisable en pratique, quitte à laisser de côté les configurations gênantes).

### 1.1.1 *Classes combinatoires et séries génératrices*

Une classe combinatoire est un ensemble $\mathcal{C}$ muni d'une application taille notée $|\cdot| : \mathcal{C} \to \mathbb{N}$ et tel que l'ensemble $\mathcal{C}_n = \{\gamma \in \mathcal{C}; |\gamma| = n\}$ des objets de taille $n$ soit fini pour tout $n \geqslant 0$.

Si on note $C_n = \mathrm{Card}(\mathcal{C}_n)$, la série génératrice $C(z)$, dite « ordinaire », associée à la classe combinatoire $\mathcal{C}$ est la série formelle

$$C(z) = \sum_{\gamma \in \mathcal{C}} z^{|\gamma|} = \sum_{n \geqslant 0} C_n z^n.$$

L'étude des séries génératrices de langages fait plus naturellement intervenir des séries génératrices ordinaires. Cependant une autre forme plus adaptée aux structures dites étiquetées existe : la série génératrice exponentielle [2] $\widehat{C}(z)$ associée à la classe $\mathcal{C}$ est

$$\widehat{C}(z) = \sum_{\gamma \in \mathcal{C}} \frac{z^{|\gamma|}}{|\gamma|!} = \sum_{n \geqslant 0} \frac{C_n}{n!} z^n.$$

On utilise abondamment, pour une série formelle $f(z) = \sum_{n \geqslant 0} f_n z$, la notation $[z^n]f(z) = f_n$ correspondant au coefficient de $z^n$.

Les séries qui ont été présentées sont univariées et leur étude permet seulement d'énumérer les classes d'objets. L'analyse d'algorithme vise plus souvent à analyser un paramètre $c$ en fonction de la taille de l'objet. Les séries génératrices bivariées ont un pouvoir d'expression plus grand puisqu'elles prennent en compte deux paramètres conjointement (l'un d'eux étant la taille). En notant $\mathcal{C}_{n,k} = \{\gamma \in \mathcal{C}_n; c(\gamma) = k\}$

1. Une vision différente des données pour les algorithmes de tri sera présentée dans le chapitre 4.
2. Des séries qui en toute logique auraient pu s'appeler extraordinaires.

et $C_{n,k} = \operatorname{Card} \mathcal{C}_{n,k}$, on obtient alors des séries[3] bivariées à deux variables

$$C(z,u) = \sum_{\gamma \in \mathcal{C}} z^{|\gamma|} u^{c(\gamma)} = \sum_{n,k \geqslant 0} C_{n,k} z^n u^k.$$

Remarquons qu'on a toujours $C(z) = C(z,1)$. Cette série génératrice donne accès à l'espérance du paramètre $c$ sur les données de taille $n$ (ici en supposant une distribution uniforme sur $\mathcal{C}_n$) à travers la formule classique utilisant une dérivation par rapport à la variable $u$

$$\mathbf{E}[c_n] = \sum_{\gamma \in \mathcal{C}_n} P(\gamma) c(\gamma) = \frac{1}{C_n} \sum_{k \geqslant 0} k C_{n,k} = \frac{[z^n] \frac{\partial}{\partial u} C(z,u)\big|_{u=1}}{[z^n] C(z,1)}.$$

En dérivant encore, on a accès à la variance ou encore aux moments de la variable aléatoire.

### 1.1.2 *Méthode symbolique*

On dispose de dictionnaires mettant en relation constructions combinatoires et opérations sur les séries génératrices. C'est un des grands intérêts de la méthode symbolique. Une fois la structure combinatoire bien comprise, on calcule assez facilement les séries génératrices correspondantes.

Dans un souci de simplicité, je rappelle informellement le principe de la méthode symbolique uniquement pour le cas des classes combinatoires « non étiquetées », et qui sont celles qui interviendront dans la suite. On verra qu'il y a également des analogies frappantes avec la vision inductive des langages rationnels.

Les structures décomposables non étiquetées sont définies à partir de classes combinatoires et d'opérations sur ces classes :

- La classe des atomes, le plus souvent constituée d'objets de taille 1, est la classe de « base », notée traditionnellement $\mathcal{Z}$.
- La classe des éléments vides (de taille 0) notée $\mathcal{E}$. (Généralement, il n'y a qu'un seul élément vide.)
- L'union disjointe $\mathcal{C} = \mathcal{A} \oplus \mathcal{B}$ de deux classes $\mathcal{A}$ et $\mathcal{B}$, définie si $\mathcal{A} \cap \mathcal{B} = \emptyset$.
- Le produit cartésien $\mathcal{C} = \mathcal{A} \times \mathcal{B} = \{(\alpha, \beta); (\alpha, \beta) \in \mathcal{A} \times \mathcal{B}\}$ de deux classes $\mathcal{A}$ et $\mathcal{B}$.
- L'opérateur SEQ, étendant le produit cartésien à un nombre fini d'ensembles permet de construire la classe des séquences finies d'objets de la classe $\mathcal{A}$.

Il existe d'autres constructions comme par exemple les opérateurs SET, CYC ou encore MSET correspondant respectivement aux classes obtenues en considérant un ensemble d'éléments (par rapport à la séquence, on oublie l'ordre et on interdit les doublons), un cycle d'éléments (un séquence «circulaire») et un multiensemble d'éléments (les répétitions sont permises).

3. On a aussi bien sûr la version exponentielle de cette série bivariée

$$\widehat{C}(z,u) = \sum_{\gamma \in \mathcal{C}} \frac{z^{|\gamma|}}{|\gamma|!} u^{c(\gamma)} = \sum_{n,k \geqslant 0} \frac{C_{n,k}}{n!} z^n u^k.$$

On autorise aussi des définitions par des systèmes d'équations qui peuvent être récursifs, mais sous certaines conditions. En effet les classes doivent être bien définies.

On a alors un dictionnaire qui permet de traduire les constructions combinatoires en séries génératrices. Pour les constructions les plus communément rencontrées on a par exemple

| Combinatoire | série gén. |
|---|---|
| $\alpha \in \mathcal{Z}$ | $z^{\lvert\alpha\rvert}$ |
| $\varepsilon \in \mathcal{E}$ | $z^0 = 1$ |
| $\mathcal{C} = \mathcal{A} \oplus \mathcal{B}$ | $C(z) = A(z) + B(z)$ |
| $\mathcal{C} = \mathcal{A} \cdot \mathcal{B}$ | $C(z) = A(z) \cdot B(z)$ |
| $\mathcal{C} = \textsc{Seq}(\mathcal{A})$ | $C(z) = \frac{1}{1-A(z)}$ |

Ce genre de dictionnaire s'étend au cas des séries multivariées si le paramètre de coût est additif par rapport au produit cartésien : si $\gamma = (\alpha, \beta)$, $c(\gamma) = c(\alpha) + c(\beta)$.

### 1.1.3 *Combinatoire analytique*

Le point de vue de la combinatoire analytique est de regarder les séries génératrices non plus comme des séries formelles mais comme des fonctions de la variable complexe. Pourvu que la fonction soit analytique et possède un rayon de convergence strictement positif, le développement en série de Taylor en o redonne exactement les coefficients de la série. On se référera utilement à [59] pour des explications rigoureuses et détaillées.

On extrait en principe facilement le coefficient $C_n = [z^n]C(z)$ grâce à la formule de Cauchy.

**Théorème (Cauchy)**
*Soit $C(z)$ un fonction analytique dans une région $\mathcal{V}$ simplement connexe de $\mathbb{C}$ contenant l'origine et $\Gamma$ une courbe simple fermée, orientée positivement, à l'intérieur de $\mathcal{V}$, qui encercle 0. On a*

$$[z^n]C(z) = \frac{1}{2i\pi}\int_\Gamma C(z)\frac{dz}{z^{n+1}}.$$

L'intérêt de cette formule est qu'elle exprime le coefficient à l'aide d'une formule intégrale. En effet, grâce à cette formule, une étude fine des singularités (i.e., les endroits où la fonction $C(z)$ cesse d'être analytique) donne des informations sur le comportement asymptotique de la suite $(C_n)$. De plus les séries génératrices que nous rencontrons, de par leur nature combinatoire, sont très particulières. Par exemple les coefficients sont tous positifs ou nuls puisqu'ils comptent des objets, d'où l'on déduit l'existence d'une singularité dominante (de plus petit module) réelle positive (théorème de Pringsheim).

En faisant passer le contour d'intégration près de cette singularité dominante, et sous certaines conditions très générales (une seule singularité de plus petit module), on obtient des résultats qui permettent

FIGURE 1: Illustration de la notion de col à l'aide d'un stéréogramme (Single Image Random Dot Stereogram).

de relier le comportement asymptotique des coefficients avec le comportement de la fonction au voisinage de la singularité.

Un théorème de transfert très général en combinatoire analytique est le suivant

**Théorème (Transfert)**
*Sous certaines conditions, si* $\rho$ *est l'unique singularité dominante de* $C(z)$ *et que*

$$C(z) \sim_{z\to\rho} (1-\frac{z}{\rho})^{\alpha}\left(\frac{\rho}{z}\log\left(\frac{1}{1-\frac{z}{\rho}}\right)\right)^{\beta},$$

*avec* $\alpha \notin \{-1,-2,\ldots\}$*, alors*

$$[z^n]C(z) \sim \frac{\rho^{-n}}{\Gamma(\alpha)} n^{\alpha-1}(\log n)^{\beta},$$

*avec* $\Gamma$ *la fonction Gamma d'Euler qui généralise la factorielle*[4]*, et définie (entre autres) comme le prolongement analytique de* $s \mapsto \int_0^\infty t^{s-1}e^{-t}dt$*.*

Nous rencontrerons d'autres transformations intégrales (comme la transformée de Mellin ou la formule de Rice) qui permettent d'exprimer des propriétés asymptotiques des coefficients des fonctions considérées.

Il arrive également que la nature des singularités rencontrée ne permettent pas d'appliquer la formule de Cauchy. C'est le cas notamment lorsque au voisinage de la singularité la fonction croît de manière exponentielle. On a alors recours à une analyse de méthode de col[5] pour obtenir l'asymptotique des coefficients.

### 1.1.4 *Analyse dynamique*

Principalement développée dans la dernière décennie par Brigitte Vallée, l'analyse dynamique a d'abord servi à l'étude d'algorithmes arithmétiques comme les algorithmes d'Euclide et de Gauss, puis a permis de modéliser une classe de sources de symboles : les sources dynamiques. Par analogie avec la physique statistique l'exécution d'un algorithme ou la production d'un mot sont vus comme une trajectoire dans un système dynamique. Les opérateurs de transfert «remplacent» les séries génératrices, on parle d'opérateurs générateurs. L'analyse complexe est remplacée par l'analyse fonctionnelle et la valeur propre dominante de l'opérateur joue le rôle de la singularité dominante. Les techniques utilisées sont donc plutôt celles

4. On a $\Gamma(n+1) = n!$.

5. Par analogie avec un col permettant de passer entre deux montagnes **??**, si on regarde le module d'une fonction complexe comme un «paysage», il est souvent judicieux d'estimer une intégrale sur un contour passant par les cols pour en tirer une information.

de l'analyse fonctionnelle, et font intervenir des opérateurs de transfert [29, 106], des opérateurs générateurs et des séries de type Dirichlet.

Les propriétés des opérateurs de transfert sont des propriétés de positivité qui, pour simplifier, s'apparentent à celles obtenues dans le théorème de Perron-Frobenius pour les matrices à coefficients positifs ou nuls, mais dans des espaces fonctionnels de dimension infinie.

Elles permettent en effet d'obtenir des résultats asymptotiques faisant intervenir la valeur propre dominante. L'article [4] utilise ce genre de technique pour fournir une analyse approfondie des algorithmes rapides de calcul de PGCD en moyenne.

## 1.2 THÉORIE DES LANGAGES

La théorie des langages a revêtu un rôle essentiel en informatique, puisqu'elle introduit un formalisme souple et puissant, bénéficiant d'un pouvoir d'expression suffisant pour bien des applications (bien que relativement faible si l'on se réfère à la hiérarchie de Chomsky [84]).

### 1.2.1 *Mots et langages.*

Le concept de mot est ici quelque peu éloigné de celui de la langue naturelle.

Nous fixons un alphabet fini ou infini (mais dénombrable) $\Sigma$ dont les éléments sont les *lettres*. Un mot (parfois appelé chaîne de caractère ou *string* en anglais) est une séquence finie de lettres. Ainsi, si l'on choisit l'alphabet latin $\Sigma = \{a, \ldots, z\}$, des séquences tels que azertyiop, zzzzz, haricot sont des mots valides.

On note $\varepsilon$ le mot vide. La longueur d'un mot $|u|$ est son nombre de lettres, et le mot vide $\varepsilon$ est de longueur zéro. On peut coller deux mots pour en former un troisième (concaténation). La longueur est ainsi un paramètre additif par rapport à la concaténation, i.e., pour $u, v \in \Sigma^*$ on a $|u \cdot v| = |u| + |v|$.

Selon la tradition en informatique et en théorie des langages, on appelle langage (formel) tout ensemble de mots. En plus des opérations ensemblistes, l'opération de concaténation s'étend naturellement aux langages. Ainsi pour deux langages $\mathcal{L}$ et $\mathcal{L}'$, on a

$$\mathcal{L} \cdot \mathcal{L}' = \{u \cdot u'; u \in \mathcal{L} \text{ et } u' \in \mathcal{L}'\}.$$

L'étoile de Kleene d'un langage est

$$\mathcal{L}^* = \bigcup_{k \geqslant 0} \mathcal{L}^k,$$

où $\mathcal{L}^0 = \{\varepsilon\}$ et $\mathcal{L}^{k+1} = \mathcal{L}^k \cdot \mathcal{L}$. On aura parfois besoin pour simplifier les notations de considérer les séquences non vides d'un langages qu'on notera

$$\mathcal{L}^+ = \bigcup_{k \geqslant 1} \mathcal{L}^k.$$

Si le mot $u \in \Sigma^*$ se décompose sous la forme $u = xyz$ avec $x, y, z \in \Sigma^*$, on dit que $x$ est un préfixe de $u$, $y$ un facteur, et $z$ un suffixe.

Dans la partie 2.1, une notion assez essentielle dans les preuves, même si cela ne sera pas détaillé, est la notion de code. En théorie des codes [33], un code est un langage $\mathcal{L}$ tel que tout mot de $\mathcal{L}^+$ admette une unique factorisation en mots de $\mathcal{L}$. Les codes les plus connus sont les codes préfixes (qui comprennent par exemple les codes de Huffman [74]). On rencontrera aussi dans la partie 2.3 des codes préfixes infinis.

### 1.2.2 *Expressions rationnelles.*

Une manière de présenter l'ensemble des expressions rationnelles $\mathcal{R}$ peut être d'utiliser les arbres d'expressions [6] qui sont des arbres binaires-unaires définis inductivement par

- le mot vide $\varepsilon$ ainsi que toute lettre $\alpha \in \Sigma$ appartiennent à $\mathcal{R}$ ;
- Si $R \in \mathcal{R}$, $\overset{*}{|}_{R} \in \mathcal{R}$ ;
- Si $R_1, R_2 \in \mathcal{R}$, $\overset{+}{/_{R_1}\backslash_{R_2}}$ et $\overset{\cdot}{/_{R_1}\backslash_{R_2}}$ sont tous deux dans $\mathcal{R}$.

Les arbres d'expressions peuvent être «aplatis» pour obtenir des expressions rationnelles (éventuellement parenthésées pour tenir compte de la précédence des opérateurs).

On peut interpréter naturellement cette construction en termes de langages en définissant inductivement une fonction $\nu$ de $\mathcal{R}$ dans $\mathcal{P}(\Sigma^*)$ telle que $\nu(\varepsilon) = \{\varepsilon\}$, $\nu(\alpha) = \{\alpha\}$ pour $\alpha \in \Sigma$, et

$$\nu(\overset{*}{|}_{R}) = R^*, \quad \nu(\overset{+}{/_{R_1}\backslash_{R_2}}) = R_1 \cup R_2 \quad \text{et} \quad \nu(\overset{\cdot}{/_{R_1}\backslash_{R_2}}) = R_1 \cdot R_2.$$

Un langage $\mathcal{L}$ est dit rationnel s'il existe une expression rationnelle $R$ telle que $\nu(R) = \mathcal{L}$. La présentation des langages rationnels ressemble étrangement à celle des classes combinatoires de la section précédente, où la classe $\mathcal{Z}$ est l'alphabet $\Sigma$, la classe vide est réduite à au mot vide $\varepsilon$, le produit cartésien correspond à la concaténation, et l'opérateur séquence à l'étoile de Kleene. La différence essentielle vient du fait que l'union est supposée disjointe pour les classes combinatoires, alors que la contrainte n'est pas imposée pour les langages rationnels.

Les expressions rationnelles ont donc une transcription en séries génératrices évidente mais seulement si l'expression décrivant le langage est non ambiguë. C'est le cas par exemple si les unions sont disjointes où s'il existe une seule façon de décomposer tout mot du langage de manière unique par rapport à l'expression régulière. On utilisera abondamment ce principe au chapitre 3.

### 1.2.3 *Automates finis*

Les langages rationnels sont également *exactement* les langages reconnus par automate fini (déterministe ou non déterministe).

6. Cette présentation m'a été inspirée par C. Nicaud. Grâce lui en soit rendue ici !

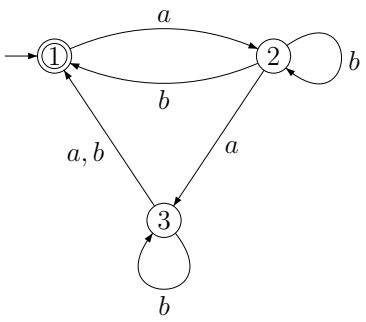



FIGURE 2: Exemple d'automate non-déterministe.

Un *automate fini* sur l'alphabet $\Sigma$ est un quadruplet $(Q, T, I, F)$, où $Q$ est un ensemble fini d'états, $T \subset Q \times \Sigma \times Q$ est l'ensemble des transitions, $I \subset Q$ est l'ensemble des états initiaux et $F \subset Q$ est l'ensemble des états finaux. On représente souvent un automate grâce à un graphe orienté où l'ensemble des sommets est $Q$ et $T$ est un ensemble d'arcs étiquetés par des lettres (voir la figure 1). Un arc étiqueté par la lettre $\alpha$ relie $p$ à $q$ si et seulement si $(p, \alpha, q) \in T$. Les états initiaux et finaux sont représentés par une marque (assez conventionnellement, respectivement, une flèche entrante ou sortante, ou encore en doublant le cercle qui représente un état final). On définit l'étiquette d'un chemin reliant deux états $p$ et $q$ comme le mot obtenu en concaténant les étiquettes du chemin. Un mot $w$ est reconnu par l'automate s'il existe un chemin reliant un état initial et un état final dont $w$ est l'étiquette. L'ensemble des mots reconnu par un automate $\mathcal{A}$, noté $\mathcal{L}(\mathcal{A})$ est le langage reconnu par $\mathcal{A}$. Un langage est dit reconnaissable s'il existe un automate qui le reconnaît.

Un automate (fini) est déterministe lorsque l'ensemble des états initiaux est réduit à un seul état et que pour tout état $p \in Q$ et toute lettre $\alpha \in \Sigma$, il existe au plus un arc partant de $p$ et étiqueté par $\alpha$. L'automate est *complet* quand il existe toujours un tel arc. Les automates déterministes sont plus simples à utiliser algorithmiquement puisque, par exemple, tout mot reconnu correspond trivialement à un unique chemin.

La présentation d'un automate déterministe est souvent légèrement différente de celle des automates vue précédemment. On considère plutôt le quadruplet $\mathcal{A} = (Q, \delta, i, F)$ avec comme différences, que $i$ désigne l'unique état initial, et que $\delta$ est une fonction partielle $\delta : Q \times \Sigma \to Q$ définie par $\delta(p, \alpha) = q$ si $(p, \alpha, q)$ est une transition. Ainsi si l'automate est complet $\delta$ est une application.

Tout automate fini *non déterministe* peut être transformé en un automate fini *déterministe* reconnaissant le même langage (au prix dans le pire des cas d'une explosion combinatoire exponentielle du nombre d'états).

Enfin, le théorème de Kleene est un théorème fondamental en théorie des langages et des automates qui fait le lien entre les deux visions précédentes. Il affirme qu'un langage est rationnel (i.e., peut être décrit par une expression rationnelle) si et seulement si il est reconnu par un automate fini.

## 1.3 SOURCES ET THÉORIE DE L'INFORMATION

Une source est un processus probabiliste qui produit un flot infini de symboles. On cherche un modèle qui s'approche le plus possible de la «réalité» tout en restant utilisable théoriquement, afin de pouvoir mener des calculs à terme.

La production de symboles est un phénomène *discret* dans le temps. On peut ainsi s'imaginer qu'à chaque coup d'horloge, un nouveau symbole est émis par la source. Le choix du symbole à émettre peut prendre en compte un grand nombre de paramètres. Nous décrivons ici plusieurs sources dont le mécanisme est probabiliste : deux sources très utilisées, les sources sans mémoire (appelées aussi sources de Bernoulli) et les chaînes de Markov, et aussi un modèle complètement général de source probabilisée.

### 1.3.1 *Sources sans mémoire*

Le principe d'une source sans mémoire est simple. On se donne un alphabet $\Sigma$ et une loi de distribution des symboles $\{p_i\}_{i\in\Sigma}$. Puis, chaque symbole $i$ est émis la probabilité $p_i$. Par définition, la source sans mémoire ne tient aucun compte des symboles précédemment émis : il n'y a aucune dépendance entre deux symboles émis par la source.

Pour la langue anglaise avec un alphabet de taille 27 (les lettres plus le caractère espace), une première tentative (grossière) de modélisation consiste à émettre chaque lettre avec une probabilité $\frac{1}{27}$. Un exemple typique d'un mot produit par une telle source est

EDBNZRBIAENHN ZUNKDMXZWHEYMHAVZWHWJZ
UFLKHYCABAOGQBQTSRDNORGCQNXWDPSTJBASDEKXHUR.

La deuxième étape, naturelle, consiste à considérer des probabilités non uniformes calculées à partir d'un corpus (ici l'intégrale de *Harry Potter* par J. K. Rowling—pour changer de Shakespeare). On obtient alors un mot qui «ressemble» déjà plus d'un point de vue syntaxique à une phrase naturelle en anglais (l'alphabet contient cette fois-ci les caractères de ponctuation «;.,!" ?'»)

O ?;RSK. ASYUS EOR FY V WIRNOH T OCFRC RTISB AEGNULINBIIE PM E VTATO"A
UDCSDHARMWMAEKHAOEIR.

### 1.3.2 *Chaînes de Markov*

Le prochain échelon à gravir consiste à prendre en compte les dépendances entre les lettres. En effet, la lettre 'T' en anglais a toutes les chances d'être suivie d'un 'H'. En français, la lettre 'Q' sera souvent suivie de la lettre 'U'. Les chaînes de Markov du premier ordre (car il est évidemment possible d'examiner les dépendances plus lointaines, non réduites à deux lettres consécutives) permettent de prendre en compte ce type de modèle.

Considérons les probabilités conditionnelles calculées à partir des fréquences de paires de lettres successives (ou digrammes)

$$p_{i|j} = p_{ji}/p_j$$

où $p_{ji}$ est la probabilité du digramme ji (la lettre j suivie de la lettre i) et $p_{i|j}$ est la probabilité conditionnelle d'avoir un symbole i lorsque le symbole précédent est j. Une chaîne de Markov est donc définie à partir d'une matrice de transition $\Pi = (p_{i|j})$ et un vecteur $\pi_0$ de probabilités initiales. Ces matrices ont beaucoup de propriétés et sont abondamment étudiées (voir [54] par exemple pour une présentation plus détaillée).

Pour générer un texte qui obéisse à ce modèle (issu d'un corpus), on peut utiliser une méthode de type Monte Carlo due à Shannon. Cette méthode permet d'éviter le calcul des probabilités à partir du corpus. On choisit un texte au hasard et on pointe une lettre aléatoirement. Supposons que cette lettre soit 'B'. On pointe à nouveau au hasard un endroit du texte et l'on parcourt le texte jusqu'à rencontrer la lettre 'B'. Le symbole à émettre est alors le symbole le suivant immédiatement. Ce symbole devient le symbole courant, et on itère le processus.

Grâce à cette méthode, on génère par exemple le texte suivant qui correspond à une chaîne de Markov du premier ordre (toujours d'après le texte de Harry Potter)

WACHT TOF ON. ATED. T AND COF CTHAR PERR. IN Y ATORON OP, HAD IVE O CONAS. HROR.

Plutôt que de prendre des dépendances entre deux lettres, on peut considérer une «fenêtre» sur les derniers caractères afin d'émettre le prochain symbole (la méthode de Shannon s'adapte facilement). On obtient alors des approximations d'ordre supérieur correspondant à des chaînes de Markov d'ordre supérieur. Par exemple, on a

NST WARRY FIVERSEST SAID HEM AS ABOG'S EVE SUDER WAL BUCHADER. AND TO OVEYES WIT.

(Chaîne de Markov deuxième ordre).

SHE GOLD A SCREAD BECAME TIME A HAD DON'S AND WITH WAITING WERE FING THE LAUGUST .

(Chaîne de Markov troisième ordre).

En faisant preuve de beaucoup d'imagination et en acceptant les néologismes au sens trouble, on peut commencer à trouver du sens à cette phrase !

### 1.3.3 *Paramétrisation d'une source de symboles*

La tâche qui consiste à modéliser une source de symboles en partant d'un texte réel (que ce soit un corpus en langue naturelle, ou encore des numéros de cartes de crédit) est ardue. Le modèle idéal tient compte de l'ensemble des caractères déjà émis pour choisir le prochain symbole, et, pour un alphabet $\Sigma$, il considère les ensembles $\{p_w\}_{w\in\Sigma^k}$ des probabilités d'apparition d'un préfixe $w$ de longueur $k$

$$p_w = P\{v \in \Sigma^{\mathbb{N}} \text{ a pour préfixe } w\}.$$

Bien sûr, on a l'égalité pour tout $k \geqslant 0$

$$\sum_{w\in\Sigma^k} p_w = 1.$$

Cette paramétrisation est basée sur le même principe que celui utilisé pour le codage arithmétique en compression. Pour chaque longueur $k$ de préfixe, les probabilités $\{p_w\}_{w\in\Sigma^k}$ définissent une partition de l'intervalle $[0,1]$ en un ensemble d'intervalles disjoints $\mathcal{I}_w = [a_w, b_w]$ défini par

$$a_w = \sum_{v\prec w} p_v, \qquad b_w = \sum_{v\preceq w} p_v = a_w + p_w.$$

Notons que l'intervalle $\mathcal{I}_w$ admet bien comme mesure $p_w$. Ces intervalles $\mathcal{I}_w$ sont appelés *intervalles fondamentaux*. De plus lorsqu'on fait grandir la taille des préfixes on obtient des raffinements successifs de l'intervalle $[0,1]$. En effet, on a dans ce modèle pour tout mot $w$ fixé

$$\sum_{\alpha\in\Sigma} p_{w\cdot\alpha} = p_w, \text{ et } \mathcal{I}_w = \bigcup_{\alpha\in\Sigma} \mathcal{I}_{w\cdot\alpha}.$$

Inversement, étant donné cette famille de probabilités fondamentales $\{p_w\}_{w\in\Sigma^*}$, on peut définir une application $M : [0,1] \to \Sigma^{\mathbb{N}}$ qui associe à tout réel $u$ de l'intervalle unité $\mathcal{I} := [0,1]$, un mot infini $M(u) := (m_1(u), m_2(u), m_3(u), \ldots) \in \Sigma^{\mathbb{N}}$. L'application $m_i$ est ici une application qui à partir d'un réel $u$ donne le $i^e$ symbole du mot $M(u)$. Pour un réel $u \in [0,1]$, le symbole $m_i(u)$ est la $i^e$ lettre (commune) à tous les préfixes $p$ de longueur supérieure ou égale à $i$ tels que $u \in \mathcal{I}_p$. Par exemple le premier symbole de $M(u)$ est obtenu en examinant les intervalles fondamentaux du premier niveau et en renvoyant la lettre qui indice l'intervalle[7] contenant $u$. Le processus est répété pour les symboles suivants en considérant les raffinements successifs des intervalles. Cette paramétrisation préserve l'ordre lexicographique sur les mots ('$\prec$') par rapport à celui sur les réels de l'intervalle $\mathcal{I}$. Ces sources générales seront abordées plus précisément (avec comme importante sous-classe les sources dynamiques [115, 17], inspirées par les systèmes dynamiques) au chapitre 4 lors de l'étude des algorithmes de tri où on considérera justement que les clefs sont produites par une telle source complètement générale.

7. En admettant que cet intervalle est unique, ce qui est vrai sauf sur l'ensemble de mesure nulle constitué des extrémités des intervalles fondamentaux.

# 2 MOTS ET LANGAGES

*Principaux articles concernés dans ce chapitre [1, 15, 19] :*
- **The standard factorization of Lyndon words : an average point of view.** Bassino, F., Clément, J., and Nicaud, C. Discrete Mathematics 290, *1 (2005), 1–25.*
- **Reverse Engineering Prefix Tables.** Clément, J., Crochemore, M., and Rindone, G. *In* Symposium on Theoretical Aspects of Computer Science (STACS) *(2009), pp. 289–300.*
- **Optimal prefix codes for pairs of geometrically-distributed random variables.** Bassino, F., Clément, J., Seroussi, G., and Viola, A. *38 pages, soumis à IEEE Transactions In Information Theory.*

*Les autres publications en rapport sont [10, 11, 13, 12, 5].*

La notion de mot, compris comme une séquence de lettres, est très simple. Dans ce chapitre, on explore plusieurs autres façons de voir les mots plus proches de la combinatoire des mots :
- **Mots de Lyndon.** Les mots sont vus circulairement, et à travers un représentant appelé mot de Lyndon.
- **Table des préfixes/suffixes.** On s'intéresse ici aux recouvrements d'un mot avec lui-même. Les meilleurs algorithmes de recherche de motifs utilisent ce type de propriétés pour accélérer les calculs.
- **Codes préfixes optimaux infinis pour une source géométrique de dimension 2.** Les mots à produire sont soumis à deux contraintes : former un code préfixe d'une part, et d'autre part que l'arbre des mots du code soit optimal par rapport aux probabilités des objets que l'on veut coder.

## 2.1 MOTS DE LYNDON

### 2.1.1 *Préliminaires*

Étant donné un alphabet $\Sigma$ totalement ordonné, un *mot de Lyndon* est un mot qui est strictement plus petit, pour l'ordre lexicographique, que tous ses conjugués (i.e., les mots obtenus en effectuant une permutation circulaire sur les positions). Par exemple, les premiers mots de Lyndon (jusqu'à la longueur 5) sur l'alphabet $\{a, b\}$ sont

$$\mathcal{L} = \{a, b, ab, aab, abb, aaab, aabb, abbb,\\ aaaab, aaabb, aabab, aabbb, ababb, abbbb, \ldots\}.$$

Les mots de Lyndon ont été introduits sous le nom de «suites standard lexicographiques» («standard lexicographic sequences») afin de calculer des bases dans les algèbres de Lie libres sur $\Sigma$ ([84], [101]). Plus précisément, on associe à un mot de Lyndon $w$ un arbre binaire $T(w)$ construit récursivement de la façon suivante : si $w$ est

une lettre, alors $T(w)$ est une feuille de l'arbre étiquetée par $w$, sinon $T(w)$ est un nœud interne ayant $T(u)$ et $T(v)$ comme fils, où $u \cdot v$ désigne la *factorisation standard* de $w$ (une factorisation telle que $u$ et $v$ soient des mots de Lyndon et que la longueur de $v$ soit maximale). À titre d'exemple on a les factorisations $aaabaab = aaab \cdot aab, aaababb = a \cdot aababb, aabaabb = aab \cdot aabb$. Cette structure encode une opération non associative, un commutateur sur le groupe libre [41], ou le crochet de Lie [84]. Les deux constructions permettent de construire des bases de l'algèbre de Lie. Les algorithmes calculant ces bases ont une complexité en moyenne déterminée en gros par la hauteur moyenne de ces arbres.

Une des propriétés remarquables de l'ensemble des mots de Lyndon est que tout mot peut se décomposer de manière unique en un produit de mots de Lyndon croissants. De plus il existe une bijection entre mots de Lyndon de longueur $n$ sur un alphabet de taille $k$ et polynômes irréductibles de degré $n$ sur $\mathbb{F}_k$ [64]. On transfère ensuite les nombreux résultats connus sur la factorisation des polynômes en produits de polynômes irréductibles pour la décomposition d'un mot quelconque en mots de Lyndon : nombre moyen de facteurs, longueur moyenne du plus long facteur [57] et du plus court [93] par exemple. Il est également assez simple de dénombrer les mots de Lyndon $\mathcal{L}_n$ de longueur $n$. En effet les mots de Lyndon peuvent être également définis comme des mots primitifs (i.e., qui ne sont pas la puissance d'un mot plus court) qui sont plus petits, pour l'ordre lexicographique, que leurs conjugués. Puisqu'un mot primitif de longueur $d$ admet $d$ conjugués distincts, et en notant $q$ le cardinal de l'alphabet, on a

$$q^n = \sum_{d|n} d \operatorname{Card}(\mathcal{L}_d).$$

Cette égalité signifie simplement que tout mot peut s'écrire comme une puissance d'un mot primitif. Les mots primitifs peuvent être partitionnés en classes de conjugaison, le représentant de la classe étant le mot de Lyndon associé. En utilisant la formule d'inversion de Möbius, on a

$$\operatorname{Card}(\mathcal{L}_n) = \frac{1}{n} \sum_{d|n} \mu(d) q^{n/d},$$

où $\mu$ est la fonction de Möbius définie sur $\mathbb{N} \setminus \{0\}$ par $\mu(1) = 1$, $\mu(n) = (-1)^i$ si $n$ est le produit de $i$ nombres premiers distincts et $\mu(n) = 0$ sinon. Par exemple, lorsque $q = 2$ (alphabet binaire), nous obtenons

$$\operatorname{Card}(\mathcal{L}_n) = \frac{2^n}{n}(1 + O(2^{-n/2})). \tag{1}$$

Plusieurs algorithmes manipulant les mots de Lyndon existent. J.-P. Duval [51] donne un algorithme qui calcule en temps linéaire la factorisation d'un mot quelconque en mots de Lyndon. Il existe également [61] un algorithme permettant d'énumérer dans l'ordre lexicographique les mots de Lyndon de longueur inférieure à un certain seuil. Chaque mot est produit à partir du précédent en temps constant en moyenne (voir [35]).

### 2.1.2 *Mots de Lyndon à facteur droit fixé*

L'objectif d'analyser l'arbre de décomposition $T(w)$ d'un mot de Lyndon est quelque peu ambitieux. Plus modestement, nous avons tenté (avec F. Bassino et C. Nicaud) d'analyser ce qui se passe à la racine de l'arbre, c'est-à-dire la première étape de la factorisation standard.

Dans un premier temps, nous avons étudié les mots de Lyndon dont le facteur droit est fixé. Nous obtenons un résultat qui nécessite de poser quelques notations. Soit $\Sigma = \{a_1 < \cdots < a_q = \gamma\}$ un alphabet totalement ordonné dont $\gamma$ désigne la lettre la plus grande. Soit $w$ un mot appartenant à $\Sigma^* \setminus \{\gamma\}^*$. Un tel mot s'écrit encore sous la forme $w = u\alpha\gamma^i$ où $u \in \Sigma^*$, $\alpha \in \Sigma \setminus \{\gamma\}$ et $i \geqslant 0$. Le successeur $S(w)$ de $w$ est défini [1] par $S(w) = u\beta$, où $\beta$ est la lettre immédiatement supérieure à $\alpha$ dans $\Sigma$. Pour tout mot de Lyndon $v$, on définit le langage

$$\mathcal{X}_\gamma = \{\gamma\} \quad \text{et} \quad \mathcal{X}_v = \{v, S(v), S^2(v), \ldots, S^{k-1}(v) = \gamma\} \quad \text{si } v \neq \gamma. \tag{2}$$

Notons le langage $\mathcal{X}_v$ est un code préfixe et que $k = 1 + q \times |v| - \sum_{i=1}^{q} i \times |v|_i$ où $q$ est la taille de l'alphabet $\Sigma$, $|v|$ est la longueur de $v$ et $|v|_i$ est le nombre d'occurrences de la $i$-e lettre de l'alphabet dans $v$.

Par exemple :

- Pour $\Sigma = \{a, b\}$, $v = aabab$ : $\mathcal{X}_{aabab} = \{aabab, aabb, ab, b\}$.
- Pour $\Sigma = \{a, b, c\}$, $v = abb$ : $\mathcal{X}_{abb} = \{abb, abc, ac, b, c\}$.

Par construction, $v$ est le plus petit élément de $\mathcal{X}_v\Sigma^*$ pour l'ordre lexicographique. Pour toute lettre $\alpha \in \Sigma$, on note $\Sigma_{\leqslant\alpha} = \{a \in \Sigma \mid a \leqslant \alpha\}$ l'ensemble des lettres inférieures à $\alpha$. On a le résultat suivant basé sur des arguments de combinatoire des mots :

**Théorème (Facteur droit fixé)**
*Soit $v$ un mot de Lyndon dont la première lettre est $\alpha$ et $u \in \Sigma^*$. Le mot $uv$ est un mot de Lyndon admettant $u \cdot v$ comme factorisation standard si et seulement si $u \in (\Sigma_{\leqslant\alpha}\mathcal{X}_v^*) \setminus \mathcal{X}_v^+$. Ainsi l'ensemble $\mathcal{F}_v$ des mots de Lyndon admettant $v$ comme facteur standard droit est un langage rationnel.*

On calcule également la série génératrice $F_v(z) = \sum_{x \in \mathcal{F}_v} z^{|x|}$ en examinant précisément la structure du langage rationnel $(\Sigma_{\leqslant\alpha}\mathcal{X}_v^*) \setminus \mathcal{X}_v^+$.

**Proposition**
*Soit $v$ un mot de Lyndon sur un alphabet de taille $q$. La série génératrice de l'ensemble*

$$\mathcal{F}_v = \{uv \in \mathcal{L} \mid u \cdot v \text{ est la factorisation standard}\}.$$

*s'écrit*

$$F_v(z) = z^{|v|}\left(1 + \frac{qz - 1}{1 - X_v(z)}\right),$$

*où $X_v(z)$ est la série génératrice associée à l'ensemble $\mathcal{X}_v$ de l'équation (2).*

1. Informellement, on prend la première lettre, partant de la droite, qui n'est pas maximale et on la remplace par la lettre suivante dans l'alphabet. On tronque le mot en ne conservant que le préfixe du mot jusqu'à la lettre modifiée. Ainsi, sur $\Sigma = \{a, b\}$, on a $S(aab\underline{a}bb) = aab\underline{b}$.

Ce résultat est plutôt surprenant car l'ensemble des mots de Lyndon n'est pas même un langage algébrique [32]. L'étude de la distribution de la factorisation standard à l'aide de cette série génératrice s'avère par contre plutôt malaisée puisqu'il faudrait pouvoir sommer sur l'ensemble des facteurs droits possibles.

### 2.1.3 *Longueur moyenne du facteur droit*

L'étude de la longueur moyenne du facteur droit en utilisant la série génératrice précédente s'avère difficile ou du moins hors de portée pour le moment. Nous avons donc changé d'approche (toujours avec F. Bassino et C. Nicaud) et proposé une description pour les mots de Lyndon qui permette l'analyse de la longueur du facteur droit à l'aide d'outils de la combinatoire analytique (et dans le cas d'un alphabet binaire pour simplifier).

La première chose à remarquer est que si $w$ est un mot de Lyndon de longueur $n-1$, le mot $aw$ est un mot de Lyndon de longueur $n$ dont la factorisation standard est $a \cdot w$. L'équation (1) nous informe donc que la longueur du facteur droit est $n-1$ pour approximativement la moitié des mots de Lyndon de longueur $n$.

Ce cas (un peu trivial) étant mis de côté, on s'intéresse à l'ensemble $\mathcal{L}_n \setminus a\mathcal{L}_{n-1}$. Si on prend un mot $u$ dans cet ensemble (de longueur supérieure ou égale à 2), il a nécessairement un préfixe de la forme $a^k b$ avec $k \geqslant 1$. On note qu'il ne peut pas y avoir de séquences de $a$ de longueur supérieure à $k$ puisque $u$ est un mot de Lyndon. On va montrer que sur cet ensemble, la longueur moyenne du facteur droit est approximativement $n/2$. Le schéma de cette preuve est le suivant :

- On définit un sous-ensemble $\mathcal{S}_n \subset \mathcal{L}_n \setminus a\mathcal{L}_{n-1}$ qui « recouvre » asymptotiquement $\mathcal{L}_n \setminus a\mathcal{L}_{n-1}$, i.e., tel que
$$\mathrm{Card}(\mathcal{L}_n \setminus a\mathcal{L}_{n-1}) = \mathrm{Card}(\mathcal{S}_n)(1 + o(1)), \tag{3}$$
- On construit une application $\varphi$ de $\mathcal{S}_n$ dans $\mathcal{S}_n$ qui est une involution (on a $\varphi(\varphi(u)) = u$), et qui a la propriété suivante : la somme des longueurs des facteurs droits de $u$ et $\varphi(u)$ est approximativement $n$. On a alors que
$$2 \sum_{u \in \mathcal{S}_n} |\mathrm{droit}(u)| = \sum_{u \in \mathcal{S}_n} (|\mathrm{droit}(u)| + |\mathrm{droit}(\varphi(u))|) \approx \mathrm{Card}(\mathcal{S}_n) \times n.$$

De là, on peut déduire que la longueur moyenne du facteur droit est $3n/4$. Les mots de Lyndon se partagent approximativement en deux : une moitié a un facteur droit de longueur $n-1$, soit une contribution de $n/2$ à la moyenne, et pour l'autre moitié, l'involution montre que la contribution à la moyenne est en $n/4$.

L'ENSEMBLE $\mathcal{S}_n$. L'ensemble $\mathcal{S}_n$ est essentiellement caractérisé en étudiant les plus longues plages de lettres $a$ consécutives dans les mots. Considérons en effet un mot $w \in (\mathcal{L} \setminus a\mathcal{L})$, il admet, si $|w| \geqslant 3$ (pour que $w$ existe), un préfixe $a^k b$ pour un certain $k > 0$. De plus ce mot admet une décomposition que nous appelons de type «max-run» qui peut être de deux types :

– L'occurrence de $a^k b$ au début du mot est unique. Mais $w$ contient nécessairement au moins une occurrence de $a^{k-1}b$ (disjointe de la première) puisque $w \notin a\mathcal{L}$. On écrit alors

$$w = a^k b w_1 \cdot a^{k-1} b w_2 \cdot a^{k-1} b w_3 \cdots a^{k-1} b w_r,$$

avec $r > 1$ et les mots $w_i$ n'ont pas d'occurrences de $a^{k-1}b$.
– Le mot $w$ possède au moins deux occurrences de $a^k b$, ce qui permet d'écrire

$$w = a^k b w_1 \cdot a^k b w_2 \cdot a^k b w_3 \cdots a^k b w_r,$$

avec $r > 1$ et les mots $w_i$ n'ont pas d'occurrences de $a^k b$.

L'intérêt de cette décomposition, est que l'on sait que la factorisation standard, si les $w_i$ sont suffisamment longs, interviendra juste avant l'un des facteurs $a^k w_i$ ou $a^{k-1} w_i$ (selon le cas). En effet le facteur droit de la décomposition est par définition un mot de Lyndon, ce qui implique qu'il commence par une plage de $a$ la plus longue possible.

L'ensemble $\mathcal{S}_n$ est constitué de tels mots en imposant trois conditions :

(i) La longueur $k$ de la plus longue plage de $a$ appartient à

$$\mathcal{I}_n = [\log_2 n - \log_2 \log_2 n - 1, 2\log_2 n[;$$

(ii) Les mots $w_i$ ont des préfixes distincts dans $\mathcal{P}_K$ (avec selon le cas $K = k-1$ ou $K = k$), où $\mathcal{P}_k$ est l'ensemble des mots $w = v_1 v_2 \cdots v_m$ avec $|w| \geqslant k$ et tels que pour $1 \leqslant i \leqslant m$, $v_i \in \mathcal{X}_k = \{b, ab, \ldots, a^{k-1}b\}$, et enfin que $|v_1 \ldots v_{m-1}| < k$ (i.e., on impose que le dernier facteur $v_m$ recouvre la position $k$).

La condition (i) permet d'assurer qu'on a une propriété du type (3). On a alors l'assurance que le nombre $r$ de plages de $a$ satisfait $r < 2\log_2 n$, ce qui permet d'obtenir que les mots $w_i$ soient suffisamment espacés dans le mot pour que (iii) puisse être vraie. La condition (iii) permet de définir l'involution $\varphi$. Cette condition paraît un peu technique, mais elle signifie simplement que les mots $w_i$ sont de longueur au moins $K$ et ont des préfixes assez longs distincts. Le fait de considérer qu'ils sont des produits de mots d'un code $\mathcal{X}_k = \{b, ab, \ldots, a^{k-1}b\}$ (correspondant à $\mathcal{X}_v$ pour $v = a^{k-1}b$) est nécessaire puisqu'on ne manipule pas des mots ordinaires mais des mots de Lyndon, et que l'on utilise assez abondamment dans les preuves la construction de cycles (Cyc) pour les séries génératrices ordinaires.

**Théorème**
*Soit $\mathcal{C}$ un code de série génératrice $C(z)$, la série génératrice des cycles primitifs d'éléments de $\mathcal{C}$ est*

$$\sum_{m \geqslant 1} \frac{\mu(m)}{m} \log\left(\frac{1}{1 - C(z^m)}\right).$$

À noter que si l'on choisit pour $\mathcal{C}$ l'alphabet $\Sigma$, on obtient la série génératrice des mots de Lyndon. Si l'on choisit $\mathcal{C} = \mathcal{X}_k$, on obtient

la série génératrice des mots de Lyndon commençant par strictement moins que $k$ lettres $a$. Enfin en prenant $\mathcal{C} = a^k b \mathcal{X}_\nu$ (pour $\nu = a^k b$), on obtient les mots de Lyndon commençant par exactement $k$ lettres $a$.

Une fois l'ensemble $\mathcal{S}_n$ ainsi caractérisé, et avant de définir l'involution $\phi$ plus précisément, on doit alors montrer que cet ensemble « recouvre » effectivement l'ensemble des mots de Lyndon appartenant à $\mathcal{L}_n \subset a\mathcal{L}_{n-1}$ comme énoncé à l'équation (3). On utilise pour cela utilisent des techniques relativement standard (bootstrapping [79], analyse de singularité [59] ou encore une majoration par borne de col).

L'INVOLUTION $\varphi$. Un mot $w$ de $\mathcal{S}_n$ admet donc une factorisation standard de la forme

$$w = a^k u u' \cdot a^K v v',$$

avec selon le cas $K = k - 1$ ou $K = k$ (selon le type de décomposition «max-run») et $u, v \in \mathcal{P}_K$. On définit alors

$$\varphi(w) = a^k u v' \cdot a^K v u',$$

et la factorisation standard est déterminée par les préfixes $u$ et $v$. On vérifie que la somme des longueurs des facteurs droit de $w$ et $\varphi(w)$ est

$$\begin{aligned} |\mathrm{droit}(w)| + |\mathrm{droit}(\varphi(w))| &= 2K + 2|v| + |u'| + |v'| \\ &= |w| + |v| - |u| + K - k \\ &= n + O(k). \end{aligned}$$

On a donc bien associé à chaque mot de $\mathcal{S}_n$ un autre mot tel que la somme des longueurs des facteurs droits soit approximativement $n$, et ce, grâce à une involution.

### 2.1.4 *Conclusion*

Pour à peu près une moitié des mots de Lyndon de longueur $n$, le facteur droit est de longueur $n - 1$. Pour ce qui concerne l'autre moitié, on observe une symétrie dans la distribution des longueurs : à chaque fois qu'on exhibe un mot dont le facteur droit est de longueur $\ell$ , on peut lui associer un mot de Lyndon dont le facteur droit est approximativement de longueur $n - \ell$ (les deux mots étant en bijection).

Récemment ces résultats ont été généralisés en utilisant une approche probabiliste [87, 40], et affinent encore un peu plus l'étude de la longueur du facteur droit. Notamment le modèle n'est pas exactement le même puisqu'on considère des probabilités pour les lettres. Le résultat principal par rapport à la factorisation standard est que la distribution de la longueur normalisée du facteur droit converge vers

$$\mu = p_1 \delta_1 + (1 - p_1) \mathbb{1}_{[0,1)},$$

où $p_1$ désigne la probabilité de la plus petite lettre. Cela signifie pour les mots qui ne sont pas dans $a\mathcal{L}_{n-1}$, la distribution $|\mathrm{droit}(w)|/n$ tend vers la distribution uniforme (ce qui implique notre résultat).

## 2.2 ALGORITHMIQUE DU TEXTE

Dans cette section, je décris essentiellement des résultats au problème inverse de la table des préfixes [15].

### 2.2.1 *Table des préfixes*

La table des préfixes d'un mot donne pour chaque position la longueur maximale du préfixe coïncidant à partir de cette position. Plus formellement [2], la table des préfixes $\mathrm{Pref}_w$ d'un mot $w \in \Sigma^+$ de longueur $n > 0$, est le tableau de taille $n$, tel que pour $0 \leqslant i < n$

$$\mathrm{Pref}_w[i] = \mathrm{lcp}(w, w[i\,..\,n-1]),$$

où lcp désigne la longueur du préfixe commun le plus long entre deux mots.

**Exemple.** Soit $w$ le mot `ababaababababa`. On a alors :

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $w[i]$ | a | b | a | b | a | a | b | a | b | a | b | a |
| $\mathrm{Pref}_w[i]$ | 12 | 0 | 3 | 0 | 1 | 5 | 0 | 5 | 0 | 3 | 0 | 1 |

Cette table contient les mêmes informations que la table des bords du mot : un bord d'un mot non vide $w$ est un facteur propre [3] $w$ qui est à la fois un suffixe et un suffixe de $w$. Ainsi le mot `aabaabaa` admet comme bords $\varepsilon$, `a`, `aa` et `aabaa`. La table des bord mémorise pour chaque position $i$ la longueur du bord maximal de $w[0\,..\,i]$. Les deux tables sont utilisées en algorithmique du texte et se calculent toutes deux en temps linéaire indépendamment de la taille de l'alphabet. Elles sont au cœur d'algorithmes efficaces de recherche de motifs (voir par exemple [73] ou [44]).

La notion duale de table des suffixes (à ne pas confondre avec le «tableau de suffixes», Suffix Array en anglais, qui désigne un tableau stockant les suffixes d'un mot en ordre lexicographique), simplifie l'étape de pré-calcul du motif pour l'algorithme de Boyer-Moore [38, 44]. La table des suffixes est ainsi essentielle dans différents algorithmes[73, 23, 47].

Le calcul de la table des bords est une question classique. Il est inspiré par un algorithme de Morris et Pratt (1970), qui est à l'origine du premier algorithme de recherche de motif en temps linéaire dans le pire des cas, et ce, indépendamment de la taille de l'alphabet [81]. Bien que la table des bords et la table des préfixes contiennent la même information sur la structure du mot, les algorithmes efficaces qui les calculent sont assez différents (voir [44, Chapter 1]). Le premier algorithme s'exprime de manière plutôt récursive contrairement au second. Les techniques pour calculer la table des préfixes utilisent ce qui a été fait sur les préfixes plus courts et se basent sur des propriétés combinatoires.

2. Dans cette section, les positions sont indexées à partir de 0, comme le veut la tradition dans le domaine.

3. Un mot $x$ est un facteur propre d'un mot $w$ s'il est distinct de $w$ (d'après [44]).

PROBLÈME INVERSE. Dans ce travail commun avec M. Crochemore et G. Rindone [15], nous avons considéré le problème inverse du calcul de la table des préfixes : tester si une table d'entiers est une table des préfixes pour un mot, et si oui, produire un tel mot.

**Exemple.** Soit $t$ et $t'$ deux tables données par :

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $t[i]$ | 6 | 0 | 0 | 2 | 0 | 1 |
| $t'[i]$ | 6 | 0 | 0 | 2 | 1 | 1 |

La table $t$ est valide puisque compatible avec le mot `abcaba` par exemple. Mais la table $t'$ n'est pas valide car il n'existe aucun mot admettant cette table de préfixes. Cependant, $t'$ est compatible jusqu'à la position 3 avec `abcaba`.

La même question peut être formulée pour d'autres structures de données du même type. En cas de réponse positive, les solutions fournissent des conditions nécessaires et suffisantes et des caractérisations complètes, ce qui est intéressant d'un point de vue fondamental. Cela est intéressant également d'un point de vue pratique par exemple pour faire de la génération aléatoire, ou pour tester une implémentation spécifique de la structure de données pour représenter de telles tables.

Sauf erreur de notre part, les tables de bords ont été les premières à être examinées sous cette optique de problème inverse par Franek et al. [60]. Leur algorithme teste un tableau d'entiers correspond à la table des bords d'un mot, et exhibe un tel mot s'il existe. Ils résolvent le problème en temps linéaire par rapport à la taille de la table, et indépendamment de la taille de l'alphabet. Un raffinement de Duval et al. [52, 53] résout le problème plus efficacement pour un alphabet de taille borné. Bannai et al. [26] étudient d'autres structures très utilisées en algorithmique du texte : le graphe acyclique orienté des sous-mots (Directed Acyclic Subsequence Graph), le graphe orienté acyclique du mot (appelé souvent DAWG ou automate minimal des suffixes), et le tableau des suffixes (Suffix Array). Les trois algorithmes de test fonctionnent également en temps linéaire.

RÉSULTATS. Le cas de la table des préfixes (ou de la table de suffixes) paraît plus difficile au premier abord, car une valeur dans la table des préfixes décrit un chevauchement au-delà de la position courante. Néanmoins, nous obtenons comme pour les autres structures étudiées un test en temps linéaire. L'algorithme peut fournir la partie initiale de la table la plus grande telle qu'elle soit compatible avec au moins une table des préfixes. Lorsque le tableau correspond à une table des préfixes, l'algorithme fournit le mot sur l'alphabet le plus petit possible pour l'ordre lexicographique. Nous montrons également que $\log_2 n$ lettres suffisent pour une table de taille $n$ dans le pire des cas. L'algorithme utilise une variable pour stocker l'ensemble des lettres qui sont interdites dans un certain contexte. Il est remarquable que l'algorithme ne nécessite pas de structure de données complexe pour représenter cette variable tout en restant en temps linéaire. Cela est dû à l'utilisation de propriétés combinatoires.

Le problème inverse lié à la table des plus longs facteurs, qui constituent un composant des méthodes de compression à la Ziv-Lempel, est une question ouverte (voir [45, 46]). On ne sait pas si une solution en temps linéaire existe.

### 2.2.2 *Conclusion*

Nous avons donné un algorithme pour vérifier qu'un tableau d'entiers est bien une table des préfixes correspondant à au moins un mot. Ce travail permet de mieux comprendre les propriétés (périodicités, recouvrements) sur les mots dont on peut éventuellement tirer parti algorithmiquement que ce soit à travers la table des préfixes ou celle des bords. Il existe aussi des variantes [75] dites paramétrées qui considèrent non plus une égalité stricte sur les mots pour calculer le plus long bord ou le préfixe commun maximal, mais une égalité à une permutation près de tout ou partie de l'alphabet (par exemple $aba \simeq bab \simeq ztz$).

Avec M. Crochemore, C. Nicaud et G. Rindone, nous avons aussi entrepris de revoir la question de l'énumération des tables des préfixes (ou des bords puisqu'elles sont équivalentes) à la lumière de nos précédents travaux. Nous cherchons plus particulièrement à déterminer l'asymptotique du nombre de tables de préfixes d'une certaine taille.

Un autre travail que je n'ai pas détaillé dans ce mémoire est lié au test d'appartenance à un code [5], et à la déterminisation d'un automate en «pétales» (travaux en lien avec ceux de F. Bassino, C. Nicaud et L. Giambruno [28]). Il y a des questions intéressantes à se poser sur une modélisation pertinente pour l'analyse en moyenne d'un ensemble de mots finis en algorithmique du texte. Mais cela reste un problème délicat car difficile à spécifier...

## 2.3 CODAGE DE SOURCE ET CODES PRÉFIXES

Dans cette section, on s'intéresse à des mots qui subissent une contrainte différente de celles qu'on a vues précédemment. On cherche à construire un ensemble de mots qui forme un code préfixe (aucun mot n'est préfixe d'un autre [4]), et optimal dans un certain sens qui va être précisé ici.

### 2.3.1 *Contexte*

Le point de vue en codage de source est différent, même si le but est toujours de construire un code.

CODAGE DE SOURCE. On se donne généralement un ensemble de symboles $\mathcal{A} \subset \mathbb{N}$ et muni d'une distribution de probabilités $p$ ($p(i)$ est la probabilité du symbole $i$). Une fonction de codage $c : \mathcal{A} \to \Sigma^*$

4. Contrairement à ce que l'appellation laisserait penser au premier abord.

associe à chaque symbole de $\mathcal{A}$ un mot fini sur l'alphabet de sortie $\Sigma$. La longueur moyenne d'un mot du code s'exprime sous la forme

$$L(\mathcal{C}) = \sum_{i \in \mathcal{A}} p(i)c(i).$$

En compression, l'objectif est alors de trouver un code $\mathcal{C}$ qui minimise cette longueur moyenne pour l'ensemble de symboles $\mathcal{A}$. On dit alors que le code est optimal. Notons au passage que parmi tous les codes optimaux par rapport à la longueur moyenne d'un mot du code, on montre qu'il y en a forcément au moins un qui est préfixe.

*Dans cette partie, on ne s'intéresse qu'aux fonctions de codages qui ont la propriété préfixe*[5] *et qui ont un alphabet de sortie binaire* $\Sigma = \{0, 1\}$. Ainsi on associe à $\mathcal{C}$ un arbre binaire dont les arêtes sont étiquetées par 0 et 1. Le code étant préfixe, les feuilles de l'arbre sont en bijection avec les symboles de $\mathcal{A}$. Le mot du code associé à un symbole se «lit» en parcourant la branche depuis la racine jusqu'à la feuille correspondante à la manière d'un trie (structure définie dans le chapitre 4). La longueur moyenne d'un mot du code se lit de plus sur l'arbre comme une longueur de cheminement (pondérée par le poids des feuilles).

CODAGE DE HUFFMAN. Le codage de Huffman [74] est une procédure qui intervient à un moment ou à un autre dans toutes les techniques de compression. Elle permet de construire un code préfixe binaire optimal pour un ensemble $\mathcal{A}$ fini de symboles muni de sa distribution de probabilité. Le principe du codage de Huffman est simple et est de type «bottom-up». On considère au départ une forêt (un ensemble d'arbres) constituée d'arbres (réduits à des feuilles) correspondant aux symboles de $\mathcal{A}$. Chaque arbre–feuille est de poids la probabilité du symbole qu'il représente. La procédure consiste ensuite en une séquence de «fusions» de nœuds. On va répéter le processus suivant : (i) on extrait (en les retirant) de la forêt deux arbres de poids les plus petits ; (ii) On crée un nœud qui sera la racine d'un arbre admettant les deux arbres précédents comme sous-arbre ; (iii) Ce nouvel arbre est introduit dans la forêt en lui attribuant un poids qui est la somme du poids de ses sous-arbres. Après avoir itéré ce processus $r - 1$ fois (où $r$ est le cardinal de $\mathcal{A}$) la forêt est réduite à un arbre préfixe dont les feuilles correspondent aux symboles de $\mathcal{A}$. Cet arbre correspond aussi à un code binaire préfixe optimal[6].

CODES DE GOLOMB. En 1966, Golomb [65] a décrit une famille de codes préfixes optimaux pour des distributions de type géométrique sur les entiers naturels, c'est-à-dire pour des distributions $p(i)$ de la forme

$$p(i) = (1 - q)q^i, \quad i \geqslant 0,$$

pour un paramètre réel $q \in [0, 1]$. Un code de Golomb de paramètre $k$ est défini de la façon suivante : pour un entier $n$ pour lequel on

5. L'image $\mathcal{C} = c(\mathcal{A})$ forme un code préfixe.

6. Du fait d'éventuels choix, par exemple s'il y a égalité au moment de choisir les deux arbres de plus petit poids, le code optimal obtenu n'est pas unique même si la longueur moyenne d'un mot du code reste la même.

pose la division euclidienne par $k$, i.e., $n = j \times k + i$ avec $0 \leqslant i < k$, et on code $c(n) = 0^j \cdot 1 \cdot Q_k(i)$, où $Q_k$ correspond à un code quasi-uniforme.( Un code quasi-uniforme $Q_k$ est simplement un code de cardinal $k$ avec $2^{\lceil \log k \rceil} - k$ mots de longueur $\lfloor \log k \rfloor$ et $2k - 2^{\lceil \log k \rceil}$ mots de longueur $\lceil \log k \rceil$. Lorsque que $k = 2^b$ est une puissance de deux, il s'agit simplement de l'écriture en binaire de $n$ sur $b$ bits [62].) Dans [62], ces *codes de Golomb* sont montrés optimaux pour toutes les distributions géométriques en choisissant le paramètre du code $k$ en fonction de $q$. Ces distributions interviennent, par exemple, pour coder des «runs lengths» (correspondant à la répétition d'un même symbole, et qui constitue également la motivation originelle de Golomb [65]), et en compression d'image pour encoder des erreurs résiduelles bien modélisées par des distributions géométriques bidirectionnelles (*two-sided geometric distributions*). Les codes optimaux pour ces derniers sont caractérisés dans [89], et sont des combinaisons ou des variantes de codes de Golomb. Les codes de Golomb ont l'avantage pratique d'encoder le symbole $i$ grâce à un calcul explicite utilisant la valeur $i$, sans recourir à des structures de données complexes ou des tables. En pratique, ils ont ainsi été adoptés dans nombre d'applications (cf. [102],[117]).

Quoiqu'il en soit, un codage symbole par symbole, peut entraîner une redondance[7] significative par rapport à l'entropie de la distribution (qui est une borne inférieure théorique), même dans le cas d'une séquence de variables aléatoires indépendantes de même loi. Une manière de s'accommoder de ce problème, tout en gardant la simplicité et une latence faible pour le codage et le décodage, est de considérer des blocs courts de $d>1$ symboles, et d'utiliser un code préfixe pour ces blocs. Avec F. Bassino, G. Seroussi et A. Viola, nous avons étudié les codes préfixes optimaux pour des paires (i.e., des blocs de taille 2) de variables aléatoires indépendantes, et identiquement distribuées. Plus précisément nous avons considéré la distribution sur les paires d'entiers positifs ou nuls $(i, j)$ avec probabilités de la forme

$$P(i, j) = p(i)p(j) = (1 - q)^2 q^{i+j} \quad i, j \geqslant 0. \tag{4}$$

Cette distribution est dite «géométrique de dimension 2» (*two-dimensional geometric distribution* – TDGD)). Elle est définie sur l'alphabet des paires d'entiers $\mathcal{A} = \{ (i, j) \mid i, j \geqslant 0 \}$. Par souci de brièveté, on désigne une telle distribution de paramètre $q$ sous l'appellation $\text{TDGD}(q)$.

Mis à part l'intérêt pratique, le problème présente également un intérêt combinatoire. Il a été prouvé dans [83] (voir également [78]) que, si l'entropie[8] $-\sum_{i \geqslant 0} P(i) \log P(i)$ d'une distribution sur les entiers positifs ou nuls est finie, alors il existe des codes optimaux qui peuvent être obtenus, à la limite, en considérant une séquence de codes de Huffman pour des versions tronquées de l'alphabet. Cependant, la preuve ne donne pas de méthode générale pour construire

7. Écart entre la valeur optimale théorique, donnée par l'entropie, et la longueur moyenne d'un mot du code.

8. $\log x$ et $\ln x$ désignent conformément à l'usage dans ce domaine, respectivement, le logarithme en base 2 et le logarithme népérien, naturel, de $x$.

de tels codes optimaux, et de fait, il existe peu de familles de distributions pour des alphabets dénombrables pour lesquelles une méthode de construction effective est connue [21][63]. Une approche algorithmique de tels codes est présentée dans [63], qui recouvre les distributions géométriques et des généralisations. Cependant cette approche ne s'applique pas au cas des TDGD, comme cela est explicitement mentionné dans [63], et, pour autant que nous le sachions, aucune méthode de construction générale pour les TDGD n'a été présentée dans la littérature (à part le cas trivial $q = \frac{1}{2}$, voir également [24]).

#### 2.3.1.1 *Définitions*

Nous souhaitons construire un code binaire préfixe C optimal pour les symboles de $\mathcal{A}$, paires $(i, j)$ d'entiers positifs ou nuls ($i, j \geqslant 0$) avec la distribution TDGD. Comme dans le cas fini de symboles, on associe à un code préfixe C un arbre binaire infini, dont les feuilles sont en bijection avec les symboles de $\mathcal{A}$, et dont chaque arête est étiquetée par 0 ou 1. La *profondeur* d'un nœud $x$ dans un arbre T, noté $\mathrm{depth}_T(x)$, est le nombre d'arêtes sur le chemin de la racine à $x$. Un *niveau* de T est l'ensemble des nœuds à une profondeur donnée $\ell$ (cet ensemble est désigné par le *niveau* $\ell$). On définit aussi le profil de l'arbre comme étant la suite $(n_\ell^T)_{\ell \geqslant 0}$ du nombre de feuilles à chaque niveau $\ell$ de T. Deux arbres sont considérés comme *équivalents* si leurs profils sont identiques. Ainsi, pour un code C, nous sommes essentiellement intéressés par le profil de l'arbre correspondant, ou encore par la *distribution des longueurs* des mots du code. Étant donné le profil de l'arbre, et un ordre sur $\mathcal{A}$ par probabilités décroissantes, il est toujours possible de définir un arbre canonique (par exemple, en assignant les feuilles en ordre lexicographique) qui définit de manière unique un code pour $\mathcal{A}$ (appelé canonique). Toutefois dans la suite, dans un souci de simplification, nous ne distinguerons pas le code de l'arbre (pas plus d'ailleurs que de son profil). Tous les arbres considérés ici sont binaires et localement complets, i.e., chaque nœud est soit une feuille, soit possède deux fils[9]. Dans la suite le terme de *sous-arbre* désigne un sous arbre entier de T, i.e., un sous-arbre constituée d'un nœud et tous ses descendants dans T.

Nous appelons $s(i, j) = i + j$ la *signature* de $(i, j) \in \mathcal{A}$. Pour une signature $s = s(i, j)$, il y a $s+1$ paires de signature $s$, toutes avec la même probabilité, $P(s)=(1-q)^2 q^s$, avec la distribution (4) sur $\mathcal{A}$. Étant donné un code C, les symboles avec la même signature peuvent être permutés sans affecter les propriétés qui nous intéressent (e.g., la longueur moyenne du code). Ainsi pour simplifier, on peut ramener la correspondance entre feuilles et symboles de $\mathcal{A}$ à celle entre feuilles et éléments du *multiensemble*

$$\hat{\mathcal{A}} = \{0, 1, 1, 2, 2, 2, \ldots, \underbrace{s, \ldots, s}_{s+1 \text{ fois}}, \ldots\}. \tag{5}$$

9. La terminologie «familiale» usuelle pour les arbres est utilisée ici : les nœuds ont des enfants, des parents, des ancêtres et des descendants. J'utilise également la convention «informaticienne» pour la visualisation des arbres, avec la racine en haut et les feuilles en bas. Ainsi les ancêtres sont en «haut» et les descendants en «bas».

En construisant l'arbre, on ne distinguera pas les différentes occurrences d'une signature $s$ ; Pour encoder ces symboles, les $s+1$ feuilles étiquetées avec $s$ seront associées aux symboles $(0, s), (1, s-1), \ldots, (s, 0)$ dans un ordre fixé arbitraire. Dans la suite, on oubliera souvent le facteur de normalisation des probabilités $P(s)$ (dans les cas où cela n'a pas d'importance), et nous utiliserons à la place des *poids* $w(s) = q^s$.

Soit un arbre (ou un code) $T$ pour $\mathcal{A}$. Soit $U$ un sous-arbre de $T$, et $s(x)$ la signature associée à une feuille $x$ de $U$. On désigne par $F(U)$ l'ensemble des feuilles de $U$, appelée *bordure* (*fringe* en anglais). Nous définissons le *poids*, $w_q(U)$, de $U$ comme

$$w_q(U) = \sum_{x \in F(U)} q^{s(x)},$$

et le *coût*, $\mathcal{L}_q(U)$, de $U$ comme

$$\mathcal{L}_q(U) = \sum_{x \in F(U)} \mathrm{depth}_U(x) q^{s(x)}$$

(l'indice $q$ sera omis quand le contexte sera clair). Quand $U = T$, nous avons $w_q(T) = (1-q)^{-2}$, et $(1-q)^2 \mathcal{L}_q(T)$ et la longueur moyenne du code $T$. Un arbre $T$ est *optimal* pour TDGD$(q)$ si $\mathcal{L}_q(T) \leqslant \mathcal{L}_q(T')$ pour tout arbre $T'$ représentant $\mathcal{A}$.

### 2.3.2 *Méthode*

Nous recourrons principalement, afin de prouver l'optimalité de codes infinis pour les TDGD, à la méthode due à Gallager et Van Voorhis [62], qui est résumée brièvement ci-dessous, adaptée à notre terminologie.

- On définit une suite d'*alphabets réduits* finis $(\mathcal{S}_s)_{s=-1}^{\infty}$. L'alphabet réduit $\mathcal{S}_s$ est un multiensemble contenant les signatures $0, 1, \ldots, s$ (avec les mêmes multiplicités que dans (5)), et un nombre fini de sous-ensembles (eux potentiellement infinis) de $\hat{\mathcal{A}}$, désignés comme *symboles virtuels*, qui forment une partition du multiensemble des signatures strictement supérieurs à $s$. Nous associons naturellement à chaque symbole virtuel un poids égal à la somme des poids des signatures qu'il contient.
- On vérifie ensuite que la suite $(\mathcal{S}_s)_{s=-1}^{\infty}$ est compatible avec la procédure de construction de Huffman (de type «bottom-up»). Cela signifie qu'après un certain nombre d'étapes de fusion de l'algorithme de Huffman sur l'alphabet (fini) réduit $\mathcal{S}_s$, on obtient $\mathcal{S}_{s'}$ avec $s' < s$. On itère le processus jusqu'à $s' = -1$.
- On applique l'algorithme de Huffman à l'alphabet $\mathcal{S}_{-1}$ (composé uniquement de symboles virtuels).

Bien que la suite d'alphabets réduits $\mathcal{S}_s$ puisse être vue comme évoluant du «bas vers le haut», le code infini obtenu correspond plutôt à une suite de codes $C_s$ finis qui grandit du «haut vers le bas» avec $s$ et se déroule en inversant les étapes de fusion de la procédure de Huffman. On montre que la suite des codes $(C_s)_{s \geqslant -1}$ *converge* vers le code $C$, c'est-à-dire ici, que pour tout $i \geqslant 1$, le $i^e$ mot du code $C_s$ (dont

on trie les mots de manière consistante) devient constant lorsque $s$ croît, et est égal au $i^e$ mot du code C. Un argument de convergence sur la suite des longueurs moyennes des codes établit ensuite l'optimalité du code C.

Cette méthode a été appliquée avec succès pour caractériser des codes infinis optimaux dans [62] et [89]. La technique est simple à appliquer une fois les alphabets réduits appropriés définis. Ici la difficulté réside donc dans le fait de «deviner» pour chaque cas la structure de ces alphabets. En un certain sens on peut parler d'une procédure par amorçage (bootstrapping), on l'on doit deviner la structure du code cherché, et utiliser cette structure pour définir les alphabets réduits, qui à leur tour permettent de prouver que la structure devinée est correcte ! Nous appliquons donc la méthode de Gallager-Van Voorhis pour prouver l'optimalité des codes pour certaines familles de TDGD correspondant à des paramètres du type $q = \sqrt[k]{2}$ et $q = 1/2^k$ qui permettent de «recouvrir» ou du moins de partitionner (lorsque $k$ varie) respectivement les valeurs $q \geqslant 1/2$ et $q < 1/2$ de l'intervalle $[0, 1]$.

### 2.3.3 *Propriété de singularité pour les codes*

Certaines caractéristiques des familles de codes optimaux pour les distributions géométriques (ou des variantes) dans le cas unidimensionnel se révèlent ne plus être vérifiées dans le cas 2-dimensionnel. Plus spécifiquement, les codes décrits dans [65] et [89] correspondent à des arbres binaires de *largeur bornée* : le nombre de mots de code de n'importe quelle longueur est borné supérieurement par une constante qui dépend des paramètres du code. De plus, la famille de codes dans le cas unidimensionnel partitionne l'espace des paramètres en régions (de mesure positive), telles que toutes les distributions dans une même région partagent le même code optimal. Ces propriétés ne sont plus vraies pour les TDGD. En particulier les codes optimaux pour les TDGD sont *singuliers* : si un code $\mathcal{T}_q$ est optimal pour $\text{TDGD}(q)$, alors $\mathcal{T}_q$ *n'est pas* optimal pour $\text{TDGD}(q')$ pour n'importe quelle valeur du paramètre $q' \neq q$.

Une conséquence importante de cette propriété de singularité est que l'ensemble des codes optimaux pour $q \in [0, 1]$ est non dénombrable et qu'il est donc illusoire de vouloir donner une caractérisation compacte [10] d'un tel ensemble, comme dans le cas unidimensionnel [65, 89]. Ainsi d'un point de vue pratique, le mieux qu'on puisse faire est de construire des codes optimaux pour une famille dénombrable de paramètres.

10. Informellement, une caractérisation compacte consiste en une caractérisation avec un nombre borné de paramètres eux aussi bornés. On souhaite en général aussi que cette caractérisation s'accompagne de procédures de codage et de décodage effectives.

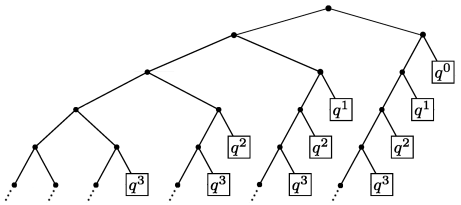



FIGURE 3: L'arbre ci-dessus décrit l'arbre optimal pour $q = 1/2$. Les feuilles sont étiquetées par leurs poids $q^s$, avec $s$ une signature.

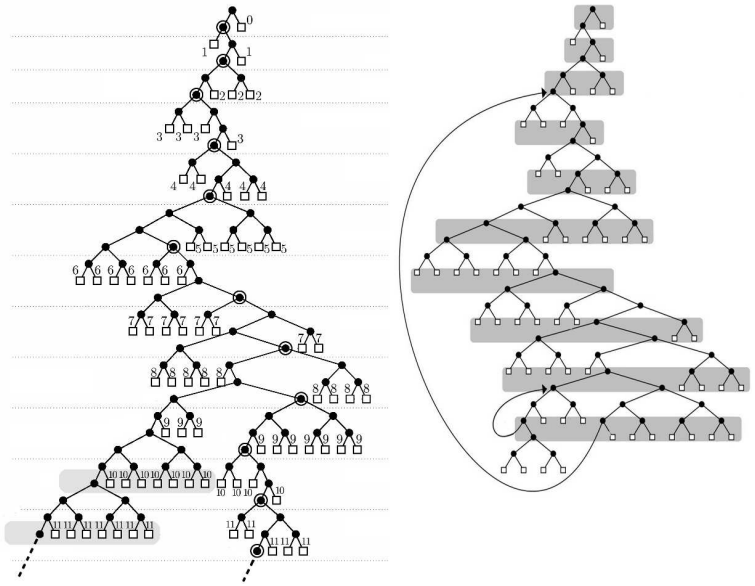


FIGURE 4: À gauche, l'arbre optimal pour les signatures $s \leqslant 11$ ($q = \frac{1}{8}$), les lignes pointillées séparent les couches (dans la partie cyclique seulement 5 types de couches). Les nœuds entourés sont des symboles virtuels utilisés dans la preuve. À droite, une représentation de l'arbre ressemblant à celle d'un automate : les mots du code sont ceux acceptés en considérant les feuilles comme états acceptants, et avec comme état initial la racine.

### 2.3.4 *Une famille de codes optimaux*

Je donne ici seulement une idée des résultats, les preuves s'avérant rapidement calculatoires et techniques, et s'appuyant toutes sur la méthode donnée précédemment. Nous avons caractérisé les codes optimaux pour des valeurs de paramètres $q = 1/2^k$ (qui donnent une partition «raisonnable» de l'intervalle $[0, 1/2]$) et $q = \sqrt[k]{1/2}$ (pour l'intervalle $[1/2, 1]$). L'arbre optimal pour $q = 1/2$ est un code de Golomb et est représenté sur la figure 2.

#### 2.3.4.1 *Les valeurs* $q = 1/2^k$

On peut décrire l'arbre (et le code) correspondant $C_{-k}$ par «couches». Après une phase initiale, les couches s'empilent de manière cyclique et fournissent une description finie du code. Les codes obtenus sont des arbres réguliers [27] : les arbres infinis correspondant aux codes ont seulement un nombre fini de sous-arbres non isomorphes. Cela permet d'avoir une représentation de l'arbre sous la forme d'un automate fini, d'avoir des expressions explicites pour la longueur moyenne

du code, et aussi des procédures de codage et de décodage effectives. Le cas $q = 1/8$ ($k = 3$) est par exemple décrit sur la figure 3. Il est aussi intéressant de noter que lorsque $k \to \infty$ ($q \to 0$), les codes optimaux $C_{-k}$ obtenus convergent vers un *code limite* $C_{-\infty}$ dans le sens suivant : un mot de code correspondant à une paire $(a, b)$ reste le même pour tout $k > k_0(a, b)$, où $k_0$ est un entier dépendant de $a$ et $b$ (ce code limite est également mentionné dans [24]). Dans tous les cas, les codes sont de largeur non bornée.

#### 2.3.4.2 *Les valeurs* $q = 2^{-1/k}$

Nous commençons par la caractérisation des codes optimaux pour $q = 2^{-1/k}$ ($k \geqslant 1$) qui est la plus proche, dans l'esprit, de celle des codes des Golomb.

En effet on a le théorème suivant :

**Théorème (Code optimal $q = 2^{-1/k}$)**
*Un code préfixe optimal $C_k$ pour* TDGD($q$)*, avec $q = 2^{-1/k}$, $k \geqslant 1$, est donné par*

$$C_k(i, j) = T_k(i \bmod k, j \bmod k) \cdot G_1(\lfloor \tfrac{i}{k} \rfloor) \cdot G_1(\left\lfloor \tfrac{j}{k} \right\rfloor), \tag{6}$$

*où $G_1$ est le code unaire (i.e., $G_1(n) = 0^n 1$), et $T_k$, que nous appelons le* code supérieur*, est un code préfixe optimal pour la source* finie

$$\hat{\mathcal{A}}_k = \{(i, j) \mid 0 \leqslant i, j < k\} \text{ avec des poids } w(i, j) = q^{i+j}\}. \tag{7}$$

Ce théorème décrit une famille de codes qui est en quelque sorte l'analogue des codes de Golomb pour les TDGD. Cependant la structure du code supérieur est bien plus complexe. Pour donner une idée de la caractérisation, le profil d'un arbre optimal est donné par le théorème suivant.

**Théorème (arbre optimal supérieur)**
*Soit $q = 2^{-1/k}$, $Q = k^2 - \lceil k(k-1)/4 \rceil$, $m = \lceil \log k^2 \rceil$, et $M = \lceil \log Q \rceil$. On définit la fonction*

$$\Delta(x) = 2k^2 - 2^{M+1} + x(x+1) - \tfrac{(k-x-2)(k-x-1)}{2}.$$

*Soit $x_0$ la racine réelle la plus grande de $\Delta(x)$, et soit $\xi = \lfloor x_0 \rfloor$. Posons*

$$(j, r) = \begin{cases} (\xi, r = \left\lfloor \tfrac{-\Delta(j)+1}{2} \right\rfloor) & \text{si } -\Delta(\xi) \leqslant 2\xi \\ (\xi + 1, 0) & \text{sinon.} \end{cases}$$

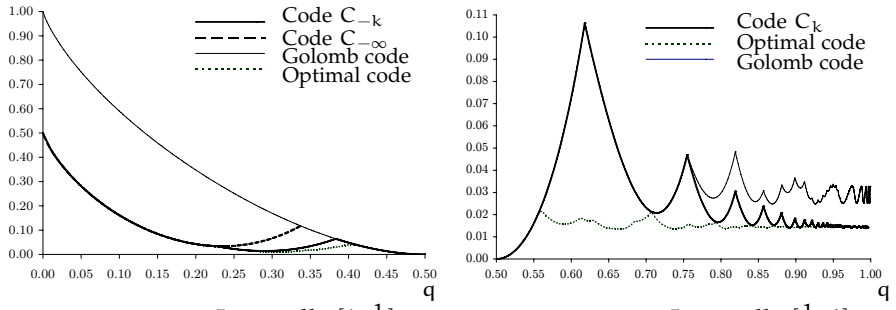
*Alors, il existe un arbre optimal pour $\hat{\mathcal{A}}_k$ défini par le profil*[11]

$$(n_{M-1}, n_M, n_{M+1}) = \Big(2^M - N + c,\ 2N - 2^M - 3c,\ 2c\Big), \tag{8}$$

*avec $c = c_k = k^2 - 2^M + \frac{j(j+1)}{2} + r$.*

On a décrit ici un code optimal (grâce à son profil (8)), mais il est évidemment possible d'obtenir tous les arbres (ou profils) optimaux (car il n'y a pas nécessairement unicité du profil non plus).

11. Le profil donne le nombre de feuilles aux niveaux $M-1, M$ et $M+1$. Pour les niveaux inférieurs $\ell < M$, $n_\ell = 0$. L'arbre étant localement complet, on reconstruit facilement tous les arbres qui ont ce profil et qui sont d'ailleurs tous optimaux.




Intervalle $[0, \frac{1}{2}]$ Intervalle $[\frac{1}{2}, 1]$

FIGURE 5: Redondance (en bits par symbole) pour le code préfixe optimal (estimé numériquement), le meilleur code de Golomb, le code limite $C_{-\infty}$, et le meilleur code $C_{-k}$ ou $C_k$ pour chaque valeur de $q$ avec à gauche, $0 < q < \frac{1}{2}$, et à droite, $\frac{1}{2} \leqslant q < 1$. Le code limite $C_{-\infty}$ est tracé jusqu'à $q = 0.33715\ldots$, point auquel la courbe intersecte celle de $C_1$.

### 2.3.5 *Redondance*

La redondance est définie comme la différence entre l'entropie par symbole et la longueur moyenne d'un mot du code. La figure 4 représente le graphe comparant les codes de Golomb (standard, symbole par symbole) avec les codes pour les TDGD pour toutes les valeurs de $q$ (les codes optimaux pour des valeurs arbitraires sont obtenus grâce à des estimations numériques avec suffisamment de précision). Nous avons également obtenu des expressions exactes pour les phénomènes oscillants observés par rapport à la redondance lorsque $q$ tend vers 1. L'étude confirme l'amélioration par rapport aux codes de Golomb, et aussi que la suite de codes choisis (ou plutôt de paramètres choisis) fournit une bonne approximation de l'ensemble des codes optimaux pour la plage $[0, 1]$ pour le paramètre $q$. La figure 4 représente la redondance de différentes familles de codes en fonction de $q$, mesuré en bits par symboles par rapport à l'entropie de la distribution géométrique [12]. Les codes concernés sont les codes préfixes optimaux (avec estimation numérique suffisante), le meilleur code de Golomb, le meilleur code $C_{-k}$ ou $C_k$ (correspondant aux familles présentées ci-dessus) pour chaque valeur de $q$ et le code limite $C_{-\infty}$. On peut se référer à l'article [19] pour plus de précisions et une explication des phénomènes oscillants observés sur la redondance.

### 2.3.6 *Conclusion*

Nous avons caractérisé des codes préfixes optimaux pour les distributions géométriques 2-dimensionnelles pour une famille de paramètres recouvrant les valeurs possibles.

Pour des sources géométriques, je souhaite m'intéresser à une classe de codes aux propriétés très intéressantes pour la synchronisation : les codes bifixes. Les mots de ces codes ont la particularité de pou-

12. L'entropie de la distribution géométrique est $H(q) = \frac{h(q)}{1-q}$, où $h(q) = -q \log q - (1-q)\log(1-q)$ est l'entropie binaire [62].

voir être décodés instantanément (i.e., dès leur lecture) quel que soit le sens de leur lecture. Cette classe donne lieu à des conjectures non élucidées depuis leur introduction, et se révèle bien plus difficile à étudier que les codes préfixes. On ne connaît pas d'analogue par exemple pour le célèbre algorithme de Huffman dans ce cas.

# 3

# MOTIFS DANS UN TEXTE ALÉATOIRE

*Principal article concerné dans ce chapitre [2] :*
**Counting occurrences for a finite set of words : combinatorial methods.**
Bassino, F., Clément, J., and Nicodème, P. Transactions in Algorithms *(2010). To appear (accepted for publication).*
*Les autres publications en rapport sont [14, 8, 9, 3].*

L'étude des statistiques d'occurrence de motifs dans un texte aléatoire permet de prédire ce qui va se passer sur un texte s'il se conforme au modèle. Souvent si le comportement observé sur un texte réel (qui peut être aussi bien un chaîne d'ADN qu'un sonnet de Shakespeare) diffère beaucoup de celui qu'on a prévu sur le modèle, on s'interroge sur la signification réelle du motif [1]. Qu'est-ce qui fait qu'il est sur-représenté ou sous-représenté ?

Par exemple en bioinformatique, un certain nombre de logiciels [113] se sont spécialisés dans la recherche de mots exceptionnels pour prédire des fonctions biologiques *In silico*. Des algorithmes de recherche de motifs sont utilisés pour trouver des candidats, et ensuite les motifs candidats sont examinés en comparant les valeurs observées avec les valeurs prédites dans le modèle (utilisation notamment du Z-score qui compare l'écart des observations en supposant une loi normale ou encore la p-valeur).

D'un point de vue fondamental, on peut s'intéresser à l'étude des statistiques d'occurrences de motifs plus ou moins complexes. Une forte demande des applications est aussi d'obtenir des formules (même approchées) qui soit efficacement calculables (voir [3, 14]).

Dans ce chapitre je me focalise sur différentes approches combinatoires qui permettent de calculer la série génératrice des occurrences.

## 3.1 CONTEXTE

On considère ici le problème du calcul effectif de la série génératrice multivariée des textes en considérant la longueur des textes et le nombre d'occurrences de mots provenant d'un ensemble fini de mots. Cet objet mathématique permet d'encoder de manière concise les statistiques d'occurrences d'un ensemble de mots dans un texte aléatoire. Plus précisément nous calculons la série génératrice $F_{\mathcal{U}}$ qui compte les textes par rapport à leur longueur et aux nombres d'occurrences de mots provenant d'un ensemble fini $\mathcal{U} = \{u_1, \ldots, u_r\}$ de $r$ mots (que nous désignerons dans la suite comme le motif). La série génératrice est nécessairement rationnelle pour des raisons expliquées plus tard, et, une fois calculée, il est relativement simple et automatisable d'ob-

1. Ou sur la pertinence du modèle, mais c'est un autre débat.

tenir toutes sortes de statistiques (voir [59] pour les façons dont on peut tirer parti des séries génératrices).

Le problème d'énumération de séquences satisfaisant certaines propriétés combinatoires a été rigoureusement formulé depuis la fin des années 70 et au début des années 80 par Goulden et Jackson [68, 69] et par Guibas et Odlyzko [71, 72].

Les premiers [68, 69] ont introduit une méthode très puissante basée sur le principe d'inclusion-exclusion pour compter les occurrences de mots appartenant à un ensemble *réduit* de mots (*i.e.*, un ensemble où aucun mot n'est facteur d'un autre) dans les textes. Cette méthode consiste à compter les textes où un sous-ensemble des occurrences sont *marquées* (on dit parfois *pointées* ou *ancrées*) et dans un second temps à retirer les redondances (un texte est compté plusieurs fois avec plusieurs marquages). Nous désignons ce principe sous la dénomination de méthode d'*inclusion-exclusion*. Goulden-Jackson ont une approche multivariée par nature où un paramètre formel est associé à chaque mot.

Les seconds [71, 72] ont introduit la notion de corrélation et d'autocorrélation, qui n'étaient pas totalement explicites dans les travaux de Goulden et Jackson. La méthode construit des systèmes d'équations sur des langages formels (non ambigus) qui se traduisent ensuite directement en termes de séries génératrices. Guibas et Odlyzko considèrent, dans le cas univarié, plusieurs problèmes comme l'énumération de textes qui évitent un motif, ou qui finissent par la première occurrence de ce motif (voir aussi [111]). Jacquet et Szpankowski [76] utilisent ces mêmes notions pour l'étude des tries et arbres des suffixes mais font le lien avec l'approche précédente en présentant des formules d'inclusion-exclusion. Enfin Régnier et Szpankowski [96] étendent les résultats de Guibas et Odlyzko au cas multivarié et comptent simultanément les occurrences de plusieurs mots mais seulement dans le cas réduit. Par la suite ces mêmes auteurs ont considéré également le cas où le texte est modélisé par une chaîne de Markov [97, 95] (voir également le livre de Szpankowski [112] et le chapitre [77] de Jacquet et Szpankowski dans le Lothaire [85]).

Une troisième approche est possible. Elle se fonde sur la théorie des automates finis. Nicodème *et al.* [92] utilisent des algorithmes classiques (1) pour construire un automate fini déterministe comportant des marques (destinées à compter les occurrences) et reconnaissant par exemple une expression régulière et (2) traduire cet automate en termes de séries génératrices grâce à l'algorithme de Chomsky-Schützenberger [42]. Cela résulte en une série génératrice bivariée qui compte les occurrences. Une variation (simple mais relativement coûteuse en espace) étend ce résultat à un source markovienne. Ce résultat s'applique immédiatement à notre problème car un ensemble fini de mots constitue un langage rationnel (et admet une expression régulière triviale). Nicodème [90] a étendu cette approche au cas multivarié et en autorisant des erreurs sur les mots de l'ensemble [2]. J. Bourdon et B. Vallée [36, 37] ont également considéré le cas des

2. Ces algorithmes sont implantés dans le package `regexpcount` de la bibliothèque `algolib` (projet Algorithmes, INRIA.

sources dynamiques. Notons, enfin, que considérant des ensembles finis de mots, la construction (1) de l'automate peut être faite directement avec l'automate de Aho-Corasick, spécifiquement dédié à la recherche de motif pour plusieurs mots. Les méthodes basées sur les automates sont, de fait, largement les plus versatiles, mais elles ont le défaut de «cacher» la structure du motif dans l'automate. Il paraît difficile d'obtenir des formules à l'aide d'automates telles que celles fournies par la méthode d'inclusion-exclusion par exemple (voir les formules obtenues à la fin de ce chapitre).

**Remarque.** Je mentionne ici principalement les approches combinatoires au problème. Prum *et al.* [94], Reinert et Schbath [98], Reinert *et al.* [99], Roquain et Schbath [104] utilisent une approche plus probabiliste pour le même type de questions (voir aussi le chapitre [100] dans [85]).

Historiquement, l'ensemble des mots dont on souhaite obtenir les statistiques d'occurrences est de deux sortes. Le cas «réduit» correspond au cas où aucun mot ne peut être facteur d'un autre, ce qui simplifie l'analyse. Le cas non réduit, plus général, permet qu'un mot soit facteur d'un autre (par exemple l'ensemble $\mathcal{U}$ peut contenir $u_1 = abbababa$ et $u_2 = baba$ bien que $u_2$ soit un facteur de $u_1$). Remarquons qu'il y a deux façons de compter les occurrences : soit c'est le nombre total d'occurrences, calculé en ajoutant pour chaque mot du motif son nombre d'occurrences, soit on compte juste le nombre de positions dans le texte signalant la fin d'un mot du motif (ce nombre est naturellement borné par la longueur du texte). Dans le cas non-réduit, le nombre total d'occurrences peut être plus grand que le nombre de positions où il y a au moins une occurrence. Au contraire dans le cas réduit, on est assuré qu'en chaque position du texte, il y a au plus une occurrence qui finit et que donc ces deux quantités sont égales.

### 3.1.1 *Notations*

Soit $\Sigma$ l'alphabet sur lequel les mots sont écrits et $\mathcal{U} = \{u_1, u_2, \ldots, u_r\}$ un ensemble fini de mots distincts sur $\Sigma$ (qu'on appellera dans la suite motif). *Par convention et pour simplifier la présentation, les mots dans un ensemble $\mathcal{U}$ seront toujours indexés pour l'ordre lexicographique croissant. Nous considérerons aussi que dans le cas d'un seul mot dans le motif $\mathcal{U} = \{u\}$, le mot $u$ est d'indice 1.*

POIDS. On dénote par $\pi(w)$ le poids du mot $w$. Le poids peut être aussi bien formel sur un monoïde commutatif (*i.e.*, $\pi(ababab) = \alpha^3\beta^3$), ou la probabilité dans le modèle de Bernoulli (aussi appelé ici source sans mémoire) $\pi(w) = \mathbf{P}(w)$ en substituant dans l'expression précédente $\alpha = \mathbf{P}(a)$ et $\beta = \mathbf{P}(b)$, ou même en posant $\pi(w) = 1$ pour le cadre énumératif.

SÉRIES GÉNÉRATRICES. On souhaite calculer ici des séries multivariées de langages. On introduit donc quelques notations. Considé-

rant un motif (décrit comme un ensemble) $\{u_1, \ldots, u_r\}$, on va assigner à chaque mot une variable formelle $x_j$ pour compter les occurrences du mot $u_j$. Étant donné un vecteur de $r$ variables formelles $\mathbf{x} = (x_1, \ldots, x_r)$ et un vecteur de $r$ entiers $\mathbf{j} = (j_1, \ldots, j_r)$, on note $\mathbf{x}^{\mathbf{j}}$ le produit $\prod_{i=1}^{r} x_i^{j_i}$. À tout ensemble de mots (ou langage) $\mathcal{L}$, on associe donc une série formelle pour représenter les statistiques

$$L(z) = \sum_{w \in \mathcal{L}} \pi(w) z^{|w|} \mathbf{x}^{\tau(w)},$$

où $\tau(w) = (|w|_1, \ldots, |w|_r)$, et $|w|_i$ est le nombre total d'occurrences de $u_i$ dans $w$ (avec possible recouvrement). Par exemple, la série génératrice associée au texte $\mathcal{L} = \{abaaabaabb\}$ avec comme motif $\mathcal{U} = \{u_1 = aa, u_2 = baa\}$ est $X(z, x_1, x_2) = z^{10}\pi(a)^6\pi(b)^4 x_1^3 x_2^2$. Plus généralement, plutôt que la série génératrice d'un texte (qui ne donne qu'un terme) on veut calculer la série génératrice multivariée $F_{\mathcal{U}}(z, \mathbf{x})$ considérant le langage de tous les textes de $\Sigma^*$ avec les occurrences des mots de $\mathcal{U}$,

$$F_{\mathcal{U}}(z, \mathbf{x}) = F(z, \mathbf{x}) := \sum_{w \in \Sigma^*} \pi(w) z^{|w|} \mathbf{x}^{\tau(w)}. \tag{9}$$

AUTOCORRÉLATION ET CORRÉLATION. Nous rappelons ici les définitions classiques de l'autocorrélation d'un mot et de la corrélation de deux mots. L'autocorrélation $\mathcal{C}_u$ d'un mot $u$ est l'ensemble

$$\mathcal{C}_u = \{h, \quad u \cdot h = y \cdot u \quad \text{avec } |y| < |u|\};$$

remarquons que le mot vide $\varepsilon$ appartient à $\mathcal{C}_u$.

De manière similaire, la corrélation du mot $u$ vers le mot $v$ est

$$\mathcal{C}_{u,v} = \{h; \quad u \cdot h = y \cdot v, \quad |y| < |u| \}.$$

L'ensemble $\mathcal{C}_{u,u}$ est l'autocorrélation du mot $u$ et si $u \neq v$ alors le mot vide $\varepsilon$ n'appartient pas à $\mathcal{C}_{u,v}$. Ces langages sont essentiels pour une approche à base de langages formels.

## 3.2 AUTOMATES

L'approche la plus versatile fait appel à la théorie des automates, et dans le cas d'un motif constitué d'un ensemble fini de mots, on a recours à la construction classique de Aho-Corasick [22, 48] qui calcule un automate reconnaissant le langage $\Sigma^*\mathcal{U}$ des mots finissant par un mot du motif. Cependant cette méthode est bien plus puissante et s'applique à tout langage régulier, et permet d'autres généralisations comme la prise en compte d'un modèle de chaîne de Markov pour la source produisant le texte (en considérant un produit d'automates) ; voir en particulier [92].

L'automate, dit de «Aho-Corasick», $\mathcal{A}_{\mathcal{U}}$ est la base de beaucoup d'algorithmes efficaces en recherche de motifs. On le décrit usuellement comme un trie (complètement) développé sur l'ensemble des

mots $\mathcal{U}$, auquel on rajoute une fonction d'échec. Soit $\mathcal{T}_{\mathcal{U}}$ le trie totalement développé représentant $\mathcal{U}$ vu comme un automate déterministe $(Q, \delta, \varepsilon, T)$, où l'ensemble des états est $Q = \mathrm{Pref}(\mathcal{U})$ (préfixes des mots de $\mathcal{U}$), l'état initial est le mot vide $\varepsilon$, l'ensemble des états terminaux est $T = \mathrm{Pref}(\mathcal{U}) \cap \Sigma^*\mathcal{U}$, et la fonction de transition $\delta$ est définie sur $\mathrm{Pref}(\mathcal{U}) \times \Sigma$ par

$$\delta(p, x) = \begin{cases} px & \text{si } px \in \mathrm{Pref}(\mathcal{U}), \\ \mathrm{Border}(px) & \text{sinon.} \end{cases}$$

Enfin la fonction d'échec Border() est définie par

$$\mathrm{Border}(v) = \begin{cases} \text{le plus long suffixe propre de } v \text{ dans } \mathrm{Pref}(\mathcal{U}) \text{ si défini,} \\ \text{ou } \varepsilon \text{ sinon.} \end{cases}$$

On identifie en général un mot $v \in \mathrm{Pref}(\mathcal{U})$ avec le nœud atteint en lisant les lettres de $v$ et en suivant les transitions dans l'arbre (vu comme un automate), si bien que Border() définit une application de l'ensemble $\mathrm{Pref}(\mathcal{U})$ sur l'ensemble des nœuds de l'arbre. On sait de plus construire efficacement une telle structure (ainsi que l'application Border()) en temps et espace $O(|\mathcal{U}|)$ [22, 48] (où $|\mathcal{U}|$ est la somme des longueurs des mots de $\mathcal{U}$) .

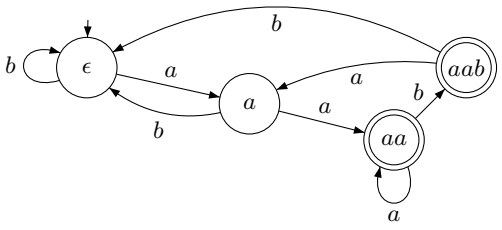
Si on ordonne les états de Aho-Corasick grâce à l'ordre lexicographique (par exemple $\varepsilon$ est l'état initial), on définit la matrice $\mathbb{T}(\mathbf{x})$ (où $\mathbf{x}$ est un vecteur de $r$ variables formelles) comme la matrice de transition de l'automate de Aho-Corasick où les variables $x_i$ marquent les états acceptant le mot $u_i$. En choisissant l'ordre sur les lignes et les colonnes pour $\mathbb{T}$ correspondant à l'ordre lexicographique sur $T$, la série génératrice s'exprime comme

$$F(z, \mathbf{x}) = \sum_{w \in \Sigma^*} \pi(w) z^{|w|} \mathbf{x}^{\tau(w)} = \begin{pmatrix} 1, & 0, & \cdots, & 0 \end{pmatrix} (\mathbb{I} - z\mathbb{T}(\mathbf{x}))^{-1} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \tag{10}$$

où $\pi(w)$ peut être vu comme le poids du mot $w$ «hérité» des coefficients de la matrice de transition. L'expression (10) implique de plus que la série est nécessairement rationnelle.

**Exemple 3.1**
*Soit* $\mathcal{U} = \{aa, aab\}$. *Toujours en ordonnant les états de l'automate selon l'ordre lexicographique pour désigner les lignes et les colonnes de la matrice, nous avons, avec* $\alpha = \pi(a)$, $\beta = \pi(b)$, *et* $\mathbf{x} = (x_1, x_2)$

$$\mathbb{T}(x_1, x_2) = \begin{pmatrix} \beta & \alpha & 0 & 0 \\ \beta & 0 & \alpha x_1 & 0 \\ 0 & 0 & \alpha x_1 & \beta x_2 \\ \beta & \alpha & 0 & 0 \end{pmatrix},$$



*et*

$$F(z, x_1, x_2) = \frac{1 - \alpha(x_1 - 1)z}{1 - z(\alpha x_1 + \beta - \alpha\beta(x_1 - 1)z + \alpha^2\beta x_1(x_2 - 1)z^2)}.$$

Les quelques exemples suivants illustrent le fait qu'on peut efficacement tirer des informations de cette série.

- Le coefficient $[z^n x_1^{n_1} x_2^{n_2}]F(z, x_1, x_2)$ est la probabilité dans le modèle de Bernoulli (où $\alpha + \beta = 1$) qu'un texte aléatoire de longueur $n$ ait $n_1$ occurrences de $aa$ et $n_2$ occurrences de $aab$.
- Dans le cadre énumératif ($\alpha = \beta = 1$), le coefficient $z^n x_1^{n_1} x_2^{n_2}$ de $F(z, x_1, x_2)$ compte le nombre de mots de longueur $n$ avec $n_1$ occurrences de $aa$ et $n_2$ occurrences de $aab$. Tout système de calcul formel peut calculer les premiers termes du développement de Taylor de cette série rationnelle. Sur notre exemple, on a

$$\begin{aligned} F(z, x_1, x_2) =& 1 + 2z + (x_1 + 3)z^2 + (x_1x_2 + x_1 + x_1^2 + 5)z^3 \\ &+ (3x_1x_2 + 2x_1 + x_1^2 + x_1^2x_2 + x_1^3 + 8)z^4 + O(z^5); \end{aligned}$$

  ce qui signifie par exemple que parmi les mots de longueur 4 (correspondant au terme en $z^4$), on a la correspondance suivante entre termes de la série, textes, et statistiques d'occurrence $\tau(w)$ de $aa$ et $aab$ dans $w$.

| Terme | Textes $w$ de longueur 4 | $\tau(w)$ |
|---|---|---|
| $3x_1x_2$ | $\{baab, aabb, aaba\}$ | $(1,1)$ |
| $2x_1$ | $\{bbaa, abaa\}$ | $(1,0)$ |
| $x_1^2$ | $\{baaa\}$ | $(2,0)$ |
| $x_1^2x_2$ | $\{aaab\}$ | $(2,1)$ |
| $x_1^3$ | $\{aaaa\}$ | $(3,0)$ |
| $8$ | $\{bbbb, bbba, bbab, babb, baba, abbb, abba, abab\}$ | $(0,0)$ |

Si on désigne par $L = \sum_{u \in \mathcal{U}} |u|$ la somme des longueurs des mots de $\mathcal{U}$, le calcul de l'automate se fait en temps et espace $O(L)$ pour un alphabet fini. L'automate a au plus $L$ états. En désignant par $N$ le nombre d'états de l'automate, la matrice de transition $\mathbb{T}$ est de taille $N^2$, mais en général la matrice est creuse puisque seulement $N \times \operatorname{Card}\Sigma$ éléments sont non nuls (l'automate est complet et déterministe avec $\operatorname{Card}\Sigma$ transitions par état). La complexité de cette approche pour le calcul de la série réside donc surtout dans le calcul du quasi-inverse de la matrice de transition.

On verra que la méthode d'inclusion-exclusion se ramène aussi au calcul d'un quasi-inverse mais cette fois-ci d'une matrice $r \times r$ (où $r$ est le nombre de mots dans $\mathcal{U}$).

Il n'est pas forcément nécessaire de calculer explicitement la série génératrice, le travail effectué dans [3] vise justement à fournir un outil permettant de «simuler» l'automate pour calculer la probabilité d'occurrence d'un motif ou encore le nombre moyen d'occurrences dans le modèle pour une certaine longueur de texte. Il convient d'être attentif avec cette approche aux problèmes de stabilité numérique.

## 3.3 LANGAGES FORMELS

Il s'agit ici d'établir un système d'équations dont les inconnues sont des langages particuliers. En un certain sens, ce n'est pas très différent de l'approche précédente par automate, qui construisait aussi un système d'équations linéaires mais où, en gros, les inconnues étaient les langages «reconnus» en considérant successivement chaque état comme final. La différence se situe dans le fait qu'on cherche à identifier les langages «utiles» et à cerner à un plus haut niveau que dans les automates les relations qu'ont entre eux les mots du motif.

Pour rendre les choses plus claires, je rappelle ici simplement l'approche de Régnier et Szpankowski [97] pour un motif constitué d'un seul mot. M. Régnier [95] a étendu par la suite cette approche en proposant une décomposition adaptée au cas d'un ensemble de mots réduit.

Pour un mot $w$, Régnier et Szpankowski [96, 97] (étendant l'approche de Guibas-Odlyzko [71, 72]) définissent une décomposition de tout texte contenant au moins une occurrence du mot $w$. Cette décomposition est non ambiguë :

1. Le début du texte jusqu'à la première occurrence est appelé *langage droit* $\mathcal{R}$ ($\mathcal{R}$ finit par une occurrence de $w$ et cette occurrence est unique) ;
2. On passe d'une occurrence de $w$ à la suivante (si elle existe) grâce à un mot du *langage minimal* $\mathcal{M}$ ;
3. La partie du texte qui suit la dernière occurrence de $w$ appartient au *langage ultime* $\mathcal{F}$.

De plus, le langage $\mathcal{N}$ comprend tous les textes sans occurrences du mot $w$. On peut alors écrire des équations de langages et obtenir la série génératrice des occurrences. On a une expression non ambiguë

$$\Sigma^* = \mathcal{N} + \mathcal{R}\mathcal{M}^*\mathcal{F},$$

et on trouve une expression pour tous les langages $\mathcal{N}, \mathcal{R}, \mathcal{M}, \mathcal{U}$ en fonction du motif $w$ et de l'autocorrélation $\mathcal{C}$. Avec un marquage approprié, on trouve la série génératrice recherchée. Cette approche s'étend au cas d'un motif réduit [96] et d'une source Markovienne [97].

Avec V. Boeva, M. Régnier et M. Vandenbogaert [14], nous avons utilisé ce cadre pour donner des algorithmes et des formules approchées pour les probabilités d'occurrences en utilisant des techniques de combinatoire analytique et en approximant les singularités des séries génératrices mises en jeu.

C'est également avec ce type d'approche que nous avons étudié avec F. Bassino, J. Fayolle et P. Nicodème la façon dont se comportent les «clumps», c'est-à-dire les portions du texte recouvertes par le motif [9]. L'étude des clumps n'avait été jusque là menée essentiellement que dans un cadre probabiliste (voir par exemple [107, 104]).

## 3.4 PRINCIPE D'INCLUSION-EXCLUSION

L'intuition derrière la méthode d'inclusion exclusion est qu'il est parfois plus difficile de spécifier un ensemble d'objets satisfaisant si-

multanément un ensemble de conditions que de décrire un ensemble d'objets qui respectent seulement un sous-ensemble de ces conditions [3].

Du point de vue des probabilités ou de la théorie des ensembles, le cœur du principe d'inclusion-exclusion réside dans l'égalité suivante pour deux ensembles $A$ et $B$

$$\mathrm{Card}(A \cup B) = \mathrm{Card}(A) + \mathrm{Card}(B) - \mathrm{Card}(A \cap B).$$

Cette égalité est généralisable et devient une somme alternée pour une famille d'ensemble $(A_i)_{1 \leqslant i \leqslant r}$. Cette formulation est parfois assez difficile à mettre en œuvre (bien que correcte !) pour des exemples complexes.

Je vais présenter à présent, une approche symbolique alternative utilisant les séries génératrices multivariées, techniquement plus simple à appliquer à notre problème.

CAMÉLIDÉS. Nous considérons un exemple élémentaire pour illustrer la méthode symbolique d'exclusion-exclusion : soit $\mathcal{P}$ l'ensemble des camélidés, constitué pour simplifier d'un chameau et d'un dromadaire. En utilisant une variable $u$ pour compter les bosses et une variable $z$ pour la taille (chaque animal est de taille 1), nous obtenons de manière standard [59]

$$\mathcal{P} = \left\{ \text{  }, \text{  } \right\}, \qquad P(z,u) = z(u+u^2).$$

Maintenant, considérons l'ensemble *distingué* $\mathcal{Q}$ défini comme l'ensemble des *objets de $\mathcal{P}$ dans lesquels chaque configuration élémentaire (ici les bosses) est soit distinguée, soit non distinguée* (on marque les bosses distinguées avec la variable $v$ et on schématise par une flèche)

$$\mathcal{Q} = \left\{ \text{  }, \text{  }, \text{  }, \text{  }, \text{  }, \text{  } \right\}$$

$$Q(z,v) = z(v+1+v^2+v+v+1) = z(2+3v+v^2) = P(z,1+v).$$

Si $Q(z,v)$ est facile à obtenir pour une raison ou une autre, alors on a $P(z,u) = Q(z,u-1)$ par simple substitution $v \mapsto u-1$. La version symbolique du principe d'inclusion est ainsi résumée : compter des objets qui contiennent des occurrences d'un «pattern» est réduit au comptage d'objets qui contiennent le «pattern» à des places distinguées (ce dernier problème étant en général plus simple).

TEXTES. Pour passer des camélidés aux textes avec leurs occurrences, on utilise la correspondance «*bosse* $\leftrightarrow$ *occurrence*». Dans le cadre des occurrences de motifs, on introduit donc la notion de textes décorés qui permet de distinguer une partie seulement des occurrences du motif. Une façon intuitive et visuelle de représenter les textes décorés est de signaler une occurrence distinguée directement

3. Une application du principe «il est plus facile d'oublier quelque chose que de se rappeler de tout»...

sur le mot au-dessus de la position correspondant à la *fin* de l'occurrence. Par exemple pour le texte $w = aababaabbbabaa$ et le motif $\mathcal{U} = \{aba\}$ on obtient la représentation quand toutes les occurrences sont distinguées

$$aab\overset{\bullet 1}{a}b\overset{\bullet 1}{a}abbbab\overset{\bullet 1}{a}a.$$

Cette représentation se généralise au cas de plusieurs mots pour $\mathcal{U}$. Pour le texte $w = abaaabaabb$ et le motif $\mathcal{U} = \{u_1 = aa, u_2 = baa\}$, on obtient, en distinguant toutes les occurrences, le mot suivant complètement décoré

$$ab\overset{\bullet 1}{\overset{\bullet 2}{a}}\overset{\bullet 1}{a}ab\overset{\bullet 1}{a}\overset{\bullet 2}{a}bb, \tag{11}$$

où ❶ et ❷ sont les indices qui signalent les positions de fin des occurrences de $u_1$ et $u_2$ respectivement ; on note que dans le cas général deux occurrences peuvent finir à la même position (ce n'est pas vrai dans le cas réduit). Les textes ne sont pas forcément complètement décorés. Pour le même texte de (11), on a aussi les textes décorés suivants possibles (entre autres)

$$ab\overset{\bullet 1}{a}a\overset{\bullet 1}{a}ab\overset{\bullet 1}{a}abb,\quad ab\overset{\bullet 1}{a}aab\overset{\bullet 1}{a}abb,\quad abaaabaabb,\quad ab\overset{\bullet 1}{\overset{\bullet 2}{a}}a\overset{\bullet 1}{\overset{\bullet 2}{a}}ab\overset{\bullet 1}{a}abb.$$

Les deux derniers textes décorés correspondent respectivement aux cas où aucune occurrence n'est distinguée et où toutes les occurrences sont distinguées (décoration complète). Naturellement avec exactement $k$ occurrences d'un motif, on obtient $2^k$ textes décorés (chaque occurrence peut ou non être distinguée). Il est important de voir que les objets combinatoires sont ici les textes décorés et que, ainsi, deux mêmes textes mais décorés différemment sont distincts.

Dans le cas simple d'un texte $\mathcal{P} = aaaa$ et d'un motif d'un seul mot $\mathcal{U} = \{u = aaa\}$, on obtient l'ensemble des textes décorés en considérant toutes les occurrences de $aaa$ dans $aaaa$

$$a\overset{\bullet 1}{\overset{\bullet 1}{a}}\overset{\bullet 1}{a}a.$$

On a aussi la série génératrice $P(z, x) = \pi(a)^4 z^4 x^2$ (où $x$ compte le nombre d'occurrences du mot $u$, et $z$ la longueur du texte). On peut décorer ce texte de quatre façons différentes

$$\{a\overset{\bullet 1}{\overset{\bullet 1}{a}}\overset{\bullet 1}{a}a, aa\overset{\bullet 1}{a}a, aaa\overset{\bullet 1}{a}, aaaa\};$$

ce qui donne pour la série génératrice des textes décorés pour $aaaa$

$$Q(z, t) = \sum_{w \in Q} \pi(w) z^{|w|} t^{\#\text{occurrences distinguées}} = \pi(a)^4 z^4 (t^2 + t + t + 1),$$

(où la variable $t$ compte les occurrences distinguées ❶ et $z$ la longueur). La relation entre $P(z, x)$ et $Q(z, t)$ est simplement (et logiquement) $Q(z, t) = P(z, 1 + t)$ puisque la substitution $x \to t + 1$ reflète le fait qu'on puisse choisir de distinguer ou non une occurrence. On peut aussi utiliser cette relation dans l'autre sens $P(z, x) = Q(z, x - 1)$.

Ce changement de variable $t \to x - 1$ est en fait très général et s'adapte sans problème au cas multivarié. Considérons le texte $\mathcal{P} = aaaaba$ et le motif $\mathcal{U} = \{u_1 = aaa, u_2 = aba\}$ ; le texte entièrement décoré (avec toutes les occurrences distinguées) est

$$a\overset{\bullet\bullet}{a}\overset{\bullet}{a}\overset{\bullet}{a}b\overset{\bullet}{a}$$

et donne la série génératrice $P(z, x_1, x_2) = \pi(a)^5\pi(b)z^6x_1^2x_2$ pour les occurrences. Il y a $2^3$ textes décorés pour le texte dans $\mathcal{P}$

$$\{aaaaba, a\overset{\bullet}{a}\overset{\bullet}{a}\overset{\bullet}{a}ba, a\overset{\bullet}{a}\overset{\bullet}{a}aba, a\overset{\bullet\bullet}{a}\overset{\bullet}{a}\overset{\bullet}{a}ba,$$
$$aaaab\overset{\bullet}{a}, aaa\overset{\bullet}{a}\overset{\bullet}{a}b\overset{\bullet}{a}, aa\overset{\bullet}{a}aab\overset{\bullet}{a}, aa\overset{\bullet}{a}\overset{\bullet}{a}ab\overset{\bullet}{a}\},$$

qui donne pour la série génératrice des textes décorés $Q(z, t_1, t_2) = \pi(a)^5\pi(b)z^6(1 + t_1 + t_1 + t_1^2 + t_2 + t_1t_2 + t_1t_2 + t_1^2t_2) = P(z, t_1 + 1, t_2 + 1)$.

Pour mettre en application le principe d'inclusion-exclusion, il reste à fournir une méthode de construction générale pour tous les textes décorés.

CLUSTERS. On définit d'abord à cette fin la notion fondamentale de *cluster*. Un cluster $c$ relativement à un motif $\mathcal{U}$ est un texte *décoré* tel que

- toutes les positions sont recouvertes par au moins une occurrence distinguée ;
- soit il y a une seule occurrence distinguée, soit toute occurrence distinguée à un recouvrement avec une autre occurrence distinguée.

Notons $C_{\mathcal{U}}$ l'ensemble des clusters possibles pour un motif $\mathcal{U}$ (ou $C$ quand le contexte est clair).

Alors, l'ensemble des textes décorés $T$ se décompose comme une séquence arbitraire de lettres de l'alphabet $\Sigma$ et de clusters

$$T = (\Sigma + C)^*. \tag{12}$$

La figure 5 illustre un cas particulier de texte décoré élément de $T$ en considérant cette décomposition. Pour appliquer la mécanique générale de la méthode symbolique [59], il est essentiel que cette décomposition soit non ambiguë : pour un texte décoré donné, il y a une unique façon de décomposer ce texte selon l'équation de langage (combinatoire) (12). Cela est vérifié car le développement du membre droit de l'équation (12) fait intervenir des unions disjointes d'ensembles : les ensembles $\Sigma$ et $C$ sont toujours disjoints en tant que textes décorés, et les clusters sont bien «délimités» car $C \cap (C \cdot C) = \emptyset$. La figure 5 souligne le fait que deux clusters peuvent être consécutifs.

Pour terminer la présentation de la méthode, considérons la série génératrice $\xi(z, \mathbf{t})$ de l'ensemble des clusters $C$,

$$\xi(z, \mathbf{t}) = \sum_{w \in C} \pi(w) z^{|w|} \mathbf{t}^{\tau(w)}, \tag{13}$$

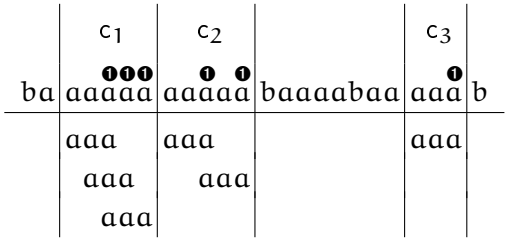



FIGURE 6: On considère le texte $w = baaaaaaaaaaabaaaabaaaaab$, le motif $\mathcal{U} = \{aaa\}$. On a représenté $w$ un texte décoré particulier avec trois clusters $c_i$ ($i = 1, 2, 3$). L'alphabet est $\Sigma = \{a, b\}$. Sur cette représentation graphique, on écrit sous le texte les occurrences distinguées pour souligner leurs recouvrements. Puisque $\mathrm{Card}(\mathcal{U}) = 1$ le symbole ❶ signale une occurrence distinguée de $u$ (l'étiquette est ici redondante).

où $\boldsymbol{\tau}(w) = (|w|_1, \ldots, |w|_r)$ et, par analogie avec la notation de (9) p. 36, $|w|_i$ désigne le nombre d'occurrences *distinguées* de $u_i$ dans $w$. Il s'ensuit des équations (12) et (13), et aussi des principes généraux [59] que la série génératrice $T(z, \mathbf{t})$ de tous les textes décorés est, avec $A(z)$ la série génératrice de l'alphabet

$$T(z, \mathbf{t}) = 1 + (A(z) + \xi(z, \mathbf{t}) + (A(z) + \xi(z, \mathbf{t}))^2 + \cdots = \frac{1}{1 - A(z) - \xi(z, \mathbf{t})}, \tag{14}$$

si bien que la série génératrice cherchée est

$$F_{\mathcal{U}}(z, \mathbf{x}) = \frac{1}{1 - A(z) - \xi(z, \mathbf{x} - \mathbf{1})}. \tag{15}$$

Ainsi on a ramené le problème du calcul de la série génératrice $F_{\mathcal{U}}(z, \mathbf{t})$ à celui du calcul de la fonction génératrice des clusters $\xi(z, \mathbf{t})$. Ce dernier problème est très simple lorsque le motif est réduit, et un peu moins dans le cas contraire.

STRUCTURE DES CLUSTERS. Si on regarde un cluster, on s'aperçoit qu'il y a deux types d'occurrences distinguées :

– les occurrences qui sont facteurs d'autres occurrences,
– les occurrences qui ne sont pas facteurs d'autres occurrences.

On définit le squelette d'un cluster, comme le cluster obtenu en supprimant toutes les occurrences facteurs. Ce squelette est défini de manière unique. Un façon d'engendrer les clusters est donc d'engendrer dans un premier temps tous les squelettes, et dans un second temps, à rajouter les occurrences «facteurs», ce qu'on peut définir plus formellement à l'aide de l'opération Flip() (voir [2]). Cette opération Flip() associe à un squelette l'ensemble de tous les clusters admettant le même squelette (en rajoutant toutes les combinaisons d'occurrences facteurs possibles).

Pour ne pas trop alourdir les notations, dans la suite, on note avec un rond plein les occurrences qui participent au squelette et avec un rond vide les occurrences qui peuvent ou non être rajoutées comme

occurrences facteurs. D'un point de vue conceptuel, cela veut simplement dire que par exemple

$$\{a\overset{①}{b}\overset{❷}{a}\} \quad \text{est équivalent à} \quad \{a\overset{❷}{b}a, \overset{❶}{a}\overset{❷}{b}a\}.$$

La subtilité réside maintenant à comprendre comment un squelette de cluster peut-être étendu, ce qui est fait grâce à la notion d'*extension droite* de deux mots. La notion d'extension droite est légèrement différente de la corrélation. L'ensemble des extensions droites d'un mot $u$ vers le mot $v$ est l'ensemble des suffixes propres $s$ non vides de $v$ tels que le mot $u \cdot s$ termine par une occurrence de $v$ et mais ne soit pas réduit à $v$.

En notant $u_1, \dots, u_r$ les squelettes avec une seule occurrence distinguée associés aux mots de $\mathcal{U}$, on peut définir la matrice $E$ des extensions droites décorées qui contient pour chaque extension droite d'un mot $u_i$ à un autre mot $u_j$ le suffixe de même longueur de $\mathrm{Flip}(u_j)$. Par exemple avec le motif non réduit $\mathcal{U} = \{aa, ab, ba, baaab\}$, on a

$$\mathrm{Flip}(u_1) = \{a\overset{❶}{a}\}, \mathrm{Flip}(u_2) = \{a\overset{❷}{b}\}, \mathrm{Flip}(u_3) = \{b\overset{❸}{a}\}, \mathrm{Flip}(u_4) = \{b\overset{③①①❹}{a}a\overset{②}{a}b\}$$

et la matrice d'extension droite est (les lignes et les colonnes sont indexées par les mots de $\mathcal{U}$)

$$\text{et} \quad E = \begin{pmatrix} \{\overset{❶}{a}\} & \{\overset{❷}{b}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{\overset{❸}{a}\} & \{\overset{③①①❹}{a}a\overset{②}{a}b\} \\ \{\overset{❶}{a}\} & \{\overset{❷}{b}\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{\overset{❸}{a}\} & \{\overset{③①①❹}{a}a\overset{②}{a}b\} \end{pmatrix}.$$

La matrice $E$ est une extension de la matrice de corrélation entre mots dans le cas non-réduit (voir [2] pour plus de précisions et le lien avec la corrélation). Nous l'avons dénommée matrice d'extension droite puisque qu'elle permet d'étendre à droite les squelettes. Chaque élément de la matrice $E$ est ainsi un suffixe du mot décoré par lequel on est en train d'étendre le squelette.

On prouve que la présentation matricielle [4] générale suivante décrit l'ensemble des clusters $C$

$$C = (\mathrm{Flip}(u_1), \dots, \mathrm{Flip}(u_r)) \cdot E^* \cdot \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon \end{pmatrix}. \tag{16}$$

La méthode symbolique nous fournit directement la correspondance en termes de série génératrice avec des règles simples

- les symboles $\alpha$ de l'alphabet : $\alpha \mapsto \pi(\alpha)z$ (poids commutatif) ;
- Les marques participant au squelette : $❶, ❷, ❸, \dots \mapsto t_1, t_2, t_3, \dots$

4. Vu de très haut, on a en fait construit une sorte d'automate qui reconnaît les clusters.

– Les marques ne participant *pas* au squelette : ①, ②, ③, ... ↦ $(1+t_1), (1+t_2), (1+t_3), \ldots$.

On obtient finalement l'expression qui traduit l'équation (16) dans le monde des séries génératrices

$$\xi(z, \mathbf{t}) = (U_1(z, \mathbf{t}), \ldots, U_r(z, \mathbf{t})) \cdot \Big(\mathbb{I} - \mathbb{E}(z, \mathbf{t})\Big)^{-1} \cdot \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \qquad (17)$$

où $U_i(z, \mathbf{t})$ est la série génératrice associée à $\mathrm{Flip}(u_i)$, et $\mathbb{E}(z, \mathbf{t})$ est une matrice de polynômes associés aux extensions droites. Ainsi toujours sur le même exemple, on a les expressions

$$U_1(z, t_1, t_2, t_3, t_4) = t_1 z^2, \quad U_2(z, t_1, t_2, t_3, t_4) = t_2 z^2,$$
$$U_3(z, t_1, t_2, t_3, t_4) = t_3 z^2, \quad U_4(z, t_1, t_2, t_3, t_4) = t_4(1+t_1)^3(1+t_2)(1+t_3) z^5,$$

et

$$\mathbb{E}(z, t_1, t_2, t_3, t_4) = \begin{pmatrix} t_1 z & t_2 z & 0 & 0 \\ 0 & 0 & t_3 z & t_4 z^4 (1+t_1)^2 (1+t_2)(1+t_3) \\ t_1 z & t_2 z & 0 & 0 \\ 0 & 0 & t_3 z & t_4 z^4 (1+t_1)^2 (1+t_2)(1+t_3) \end{pmatrix},$$

ce qui permet d'obtenir la série génératrice des clusters grâce à la formule (17).

Il ne reste plus qu'à réintroduire l'expression de $\xi(z, \mathbf{t})$ dans l'équation (15) pour finir le calcul de la série génératrice totale des occurrences pour tous les textes.

QUELQUES EXEMPLES D'APPLICATIONS. L'avantage de la formulation proposée ici est qu'elle permet de réinterpréter les décompositions pour en déduire des formules. L'application de techniques standard de la combinatoire analytique (voir [59]) à la série multivariée $F(z, \mathbf{x})$ donne accès aux statistiques d'occurrences comme la moyenne du nombre d'occurrences, la variance ou la covariance.

**Proposition (Moyenne et variance)**
*Soit $\mathcal{U} = \{u_1, \ldots, u_k\}$ un motif. La valeur moyenne et la variance de la variable aléatoire $X_n$ comptant le nombre d'occurrences de $\mathcal{U}$ dans un texte aléatoire de longueur $n$ et pour une source sans mémoire sont*

$$\mathbf{E}[X_n] = \sum_{u \in \mathcal{U}} \pi(u)(n - |u| + 1),$$
$$\frac{1}{n}\mathbf{Var}[X_n] = \pi(\mathcal{U}) - \sum_{u,v \in \mathcal{U}} \pi(u)\pi(v)(|u| + |v| - 1) + 2 \sum_{u,v \in \mathcal{U}} \pi(u)\pi(\mathcal{E}_{u,v}) + 2 \sum_{\substack{u,v \in \mathcal{U} \\ u \neq v}} \pi(u)|u|_v + o(1).$$

Notons que la dernière somme est non nulle seulement si l'ensemble des mots est non réduit, et $|u|_v$ est le nombre d'occurrences de $v$ dans $u$.

On peut aussi caractériser l'expression de la covariance pour deux motifs (pas nécessairement disjoints) $\mathcal{U}$ et $\mathcal{V}$.

**Proposition (Covariance)**
*Soit* $\mathcal{U} = \{u_1, \ldots, u_k\}$ *et* $\mathcal{V} = \{v_1, \ldots, v_j\}$ *deux motifs. La covariance des variables aléatoires* $X_n$ *et* $Y_n$ *comptant respectivement le nombre d'occurrences de* $\mathcal{U}$ *et* $\mathcal{V}$ *dans un texte aléatoire de longueur* $n$ *produit par une source sans mémoire vérifie*

$$\frac{1}{n}\,\mathbf{Cov}(X_n, Y_n) = \pi(\mathcal{U}\cap\mathcal{V}) - \sum_{u\in\mathcal{U},v\in\mathcal{V}} \pi(u)\pi(v)\big(|u|+|v|-1\big) + \sum_{u\in\mathcal{U},v\in\mathcal{V}} \Big(\pi(u)\pi(\mathcal{E}_{u,v}) + \pi(v)\pi(\mathcal{E}_{v,u})\Big) + \sum_{\substack{u\in\mathcal{U},v\in\mathcal{V}\\ u\neq v}} \Big(|u|_v\pi(u) + |v|_u\pi(v)\Big) + o(1).$$

### 3.4.1 *Conclusion*

Notre contribution se trouve dans l'application du principe d'inclusion-exclusion pour le comptage d'occurrences, dû à Goulden et Jackson (1979, 1983) mais étendu dans le cas non réduit. Noonan et Zeilberger (1999) ont fourni un package MAPLE qui prend en compte ce cas non-réduit, mais sans donner d'expressions de la série ni donner de preuves détaillées. Nous avons tenté de valider cette approche utilisant la technique d'inclusion-exclusion symbolique en donnant des preuves rigoureuses, et des exemples de formules (espérances, covariances) qu'on peut obtenir à partir des expressions de séries génératrices obtenues. Récemment Kong [82] applique l'approche de Noonan et Zeilberger pour le cas réduit et un modèle de Bernoulli (sans mémoire) asymétrique pour les symboles du texte. Il compare également cette méthode avec celle utilisée par Régnier et Szpankowski, pour mettre en avant la simplicité conceptuelle du principe d'inclusion-exclusion.

On note cependant que l'approche à base de langages formels ou d'automates fournit des informations non accessibles à la méthode d'inclusion-exclusion. Ainsi, le temps d'attente de la première occurrence ou le temps entre deux occurrences d'un même mot ou de deux mots (éventuellement en interdisant l'apparition d'autres mots). Par contre, il n'existe pas à notre connaissance de méthode générale qui résout le problème du comptage pour un motif non réduit avec une approche fondée sur des équations de langages formels.

Enfin, dans les perspectives, il reste des réflexions à mener sur la portée de la méthode d'inclusion-exclusion dans ce contexte. Est-il possible par exemple de compter non plus les occurrences (plusieurs mots du motif peuvent finir à la même position) mais plutôt les positions d'occurrences du motif (chaque position participant à au plus une occurrence) à la manière de [91] comme il est facile de le faire avec les techniques à base d'automates ?

Il reste également à élucider si une équation comme (16) ne permet pas directement de décider si pour un motif donné, on a une loi multivariée normale (voir [30, 91] pour des résultats de ce type avec d'autres approches).

# 4

# ALGORITHMES DE TRI ET DE RECHERCHE

*Principaux articles concernés dans ce chapitre [18, 6] :*
- **The number of symbol comparisons in quicksort and quickselect.** Vallée, B., Clément, J., Fill, J. A., and Flajolet, P. *In* ICALP 2009 *(2009), S. A. et al. (Eds.), Ed., vol. Part I, LNCS 5555, pp. 750–763.*
- **Dynamical sources in information theory : a general analysis of tries structures.** Clément, J., Flajolet, P., and Vallée, B. Algorithmica 29 *(2001), 307–369. (special issue)*[a].

*Les autres publications en rapport sont [7, 16, 17, 4].*

*a.* Cet article rassemble en grande partie mes résultats de thèse [20].

Tout étudiant suivant un cours de base en algorithmique apprend, qu'en moyenne, la complexité, mesurée en nombre de comparaisons, de Quicksort est $O(n \log n)$, que celle de la recherche dichotomique (binary search) est $O(\log n)$, et que celle du tri radix est $O(n \log n)$ ; voir par exemple [80, 108]. Ces affirmations se basent sur des hypothèses spécifiques — que les comparaisons entre les données (ou clefs), pour les deux premiers, et les comparaisons entre symboles pour le troisième ont un coût *unitaire* — et elles ont le mérite évident de présenter un tableau facile à assimiler de la complexité des algorithmes de tri. Cependant, ces hypothèses simplificatrices sont de fait discutables : elles ne permettent pas de comparer précisément les avantages et inconvénients d'algorithmes reposant sur des principes différents (i.e., ceux qui comparent les clefs et ceux qui utilisent une représentation sous forme de suite de symboles) d'une manière qui soit totalement satisfaisante à la fois du point de vue de la théorie de l'information et du point de vue algorithmique. En effet, d'abord le calcul n'est pas vu à son niveau le plus fondamental, c'est-à-dire, par la manipulation de symboles élémentaires tels que le bit, l'octet, le caractère ou le mot-machine. De plus les analyses simplifiées ne sont pas toujours informatives dans beaucoup d'applications (bases de données ou traitement de la langue naturelle), où l'information est de nature hautement «non atomique», au sens qu'elle ne se réduit pas à un seul mot-machine.

## 4.1 ARBRES DIGITAUX ET ABR : DEUX PARADIGMES DIFFÉRENTS

Pour fixer les idées, quand on envisage une structure de données, deux paradigmes principaux s'affrontent, avec par exemple d'un côté la structure d'arbre digital (ou trie) et de l'autre la structure d'arbre binaire de recherche (ABR).

TRIES. Les arbres digitaux, souvent appelés *tries*, sont à la fois une structure abstraite et une structure de données destinée à re-

présenter un ensemble de mots. En tant que structure abstraite, le trie correspond à construire une partition basée sur les symboles rencontrés dans les mots. Considérons un alphabet $\Sigma = \{a_1, \ldots, a_r\}$, et soit $Y \subset \Sigma^{\mathbb{N}}$ un ensemble fini de mots infinis sur $\Sigma$. Le trie associé à $Y$ est défini récursivement par la règle

$$\mathrm{trie}(Y) = \langle \mathrm{trie}(Y \setminus a_1), \ldots, \mathrm{trie}(Y \setminus a_r) \rangle,$$

où $Y \setminus \alpha$ désigne le sous-ensemble constitué à partir de $Y$ en prenant les mots commençant par $\alpha$, puis en supprimant ce symbole $\alpha$ initial. La récursion s'arrête dès que $Y$ contient moins de deux éléments. L'intérêt du trie est qu'il considère l'ensemble minimal de préfixes nécessaires pour distinguer tous les mots de $Y$ dans leur ensemble. Dans leur version abstraite, les tries peuvent donc être vus essentiellement comme des arbres préfixes de la théorie des codes (à longueur variable).

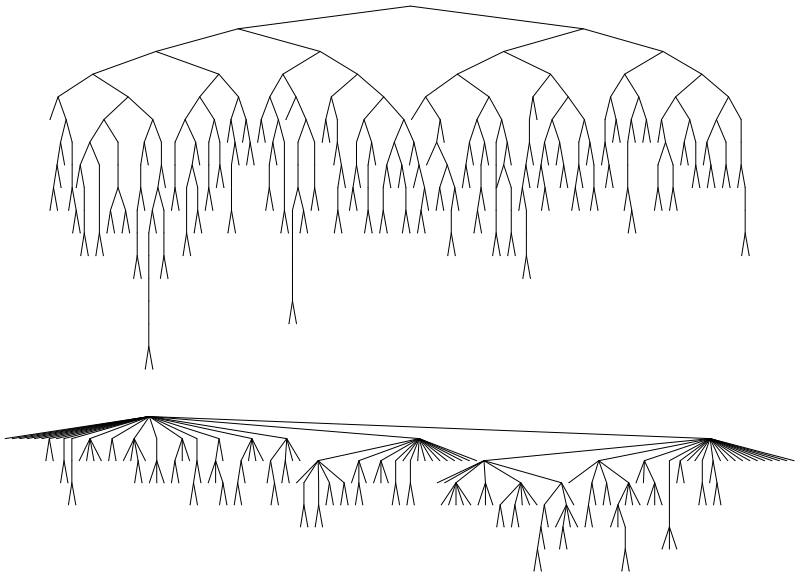


FIGURE 7: Les tries construits d'après les développements en base 2 (correspondant au modèle uniforme sur les mots binaires d'une même longueur) et en fraction continue (qui rentre dans le modèle des sources dynamiques) pour les mêmes nombres tirés uniformément sur $[0, 1]$. Le trie en fraction continue : 100 nœuds externes, hauteur 7, longueur de cheminement 505 et 75 sommets internes. Le trie binaire : 100 nœuds externes, hauteur 16, longueur de cheminement 1 323 et 222 sommets internes.

La structure d'arbre $\mathrm{trie}(Y)$ supporte évidemment la recherche d'un mot $w$ en parcourant la branche étiquetée par $w$. On peut aussi facilement implanter les procédures d'insertion, de suppression, si bien qu'on a à disposition une structure dynamique de type *dictionnaire*. De plus, les tries supportent efficacement des opérations ensemblistes comme l'union et l'intersection [114], ainsi que des opérations de recherche incomplètes (non complètement spécifiées) ou d'intervalles [103]. Des variantes adaptées en font une méthode de choix pour la manipulation du texte [67, Ch. 7]. Cette variété des applications jus-

tifie de considérer la structure de trie comme l'une des structures de données fondamentales en informatique [67, 80, 86, 110].

Pour information, on rappelle les résultats principaux de [6] pour l'analyse en moyenne d'un trie construit sur $n$ éléments, que j'ai obtenus avec P. Flajolet et B. Vallée dans le cadre général des sources probabilisées présenté dans le chapitre 1 (avec bien sûr des conditions sur la source).

(i) La valeur moyenne $H(n)$ de la hauteur d'un trie d'ordre $\log n$

$$H(n) \sim \frac{2}{|\log c(S)|} \log n,$$

où $c(S) = \lim_{k\to\infty} (\sum_{w\in\Sigma^k} p_w^2)^{1/k}$ est la probabilité de coïncidence.

(ii) La taille moyenne $S(n)$ du trie (i.e., le nombre de nœuds), à quelques minimes fluctuations près, est bien approximée par une quantité d'ordre $n$

$$S(n) \approx \frac{1}{h(S)}\, n,$$

où $h(S) = \lim_{k\to\infty} -\frac{1}{k} \sum_{w\in\Sigma^k} p_w \log p_w$ est l'entropie de la source.

(iii) La longueur de cheminement moyenne $P(n)$ du trie, qu'on peut interpréter comme le nombre moyen de comparaisons de symboles nécessaires à la construction du trie, est d'ordre $n \log n$ et fait également intervenir l'entropie

$$P(n) \sim \frac{1}{h(S)} n \log n.$$

D'un point de vue pratique, la façon dont on choisit un embranchement dans l'arbre peut être implantée de diverses manières. L'analyse de l'impact de la représentation a été faite dans [16, 6] (voir également [39] pour une très jolie analyse de la hauteur des tries ave des poids sur les arêtes). En particulier, le choix d'utiliser des arbres binaires de recherche (ABR) pour représenter les fils d'un nœud s'avère un bon compromis entre efficacité et occupation en mémoire [1] (voir [31, 43]).

ARBRES BINAIRES DE RECHERCHE. Contrairement au trie qui considère que les données (ou les clefs) à représenter sont des séquences de symboles, l'ABR considère les clefs comme des données atomiques, indivisibles. Le principe est le suivant pour construire un arbre binaire : partant d'un arbre vide, on insère la première clef à la racine. Ensuite pour chaque nouvelle clef (qu'on suppose en général distincte des précédentes), selon sa valeur, on poursuit récursivement la descente dans l'arbre en choisissant la branche gauche si la clef à insérer

1. En collaboration avec M. Archibald, j'ai fait une étude [7] concernant les ABR avec un modèle de clefs répétées. Il y a alors un choix à faire : soit insérer la première occurrence d'une clef, soit insérer toutes les occurrences en modifiant la règle d'insertion à la racine et en insérant par exemple dans le sous-arbre gauche les clefs inférieures *ou égales*. Cette étude réapparaît notamment dans l'étude des tries hybrides avec ABR aux nœuds.

est supérieure à la valeur stockée dans le nœud et à droite sinon. Enfin arrivant à un nœud vide on insère directement la clef.

Les arbres binaires de recherche sont aussi une structure fondamentale en informatique. Et de nombreuses librairies logicielles utilisent des variantes : équilibrées (AVL), sous forme d'arbres bicolores (dans la librairie JAVA notamment) ou encore avec une variante «randomisée» efficace [88].

Les analyses des arbres binaires de recherche sont très nombreuses, on peut se référer par exemple à [50].

Une autre raison pour s'intéresser aux arbres binaires de recherche est qu'on peut «lire» l'exécution de l'algorithme Quicksort sur un ABR. Grâce à l'isomorphisme bien connu entre QuickSort et la construction des arbres binaires de recherche (ABR), les résultats obtenus sur les ABR fournissent des résultats sur Quicksort. En effet au cours de l'exécution de l'algorithme Quicksort (voir les algorithmes 1 et 2 p. 52), à une étape courante on peut voir le pivot comme la racine du sous-arbre et la partition correspond aux sous-arbres gauche et droit.

**Algorithme 1** Partition$(B, k)$ : partitionne $B$ en deux parts par rapport au pivot $k$

$B_- \leftarrow \emptyset, B_+ \leftarrow \emptyset$
**for** ($i$ from $0$ to $n-1$) **do**
  **if** $B[i] < k$ **then**
    $B_- \leftarrow B_- \cup B[i]$
  **else**
    $B_+ = B_+ \cup B[i]$
  **end if**
**end for**
Return $B_-, B_+$

**Algorithme 2** QuickSort$(n, B)$ : trie un tableau $B$

Choisir aléatoirement un $k$ dans $B$
$(k, B_-, B_+) \leftarrow$ Partition$(B, k)$
QuickSort$(k-1, B_-)$
QuickSort$(n-k, B_+)$

Je rappelle ici juste le résultat de l'analyse en moyenne du nombre $C_n$ de comparaisons de clefs de Quicksort dans le modèle le plus simple (modèle uniforme sur les permutations de $\{1, \ldots, n\}$

$$\mathbf{E}[C_n] = 2(n+1)H_n - 4n = 2n \log n \frac{2(\gamma-2)}{n} + 2\log n + 2\gamma + 1 + o(1/n),$$

où $H_n = \sum_{i=1}^{n} 1/i$ et $\gamma = 0.5772\ldots$ est la constante d'Euler.

La problématique est ici de s'interroger sur la façon dont on peut comparer ce genre de résultats avec ceux obtenus sur les tries. Nous proposons de compter non plus les comparaisons de clefs mais les comparaisons élémentaires de symboles en considérant que les clefs sont une suite de symboles.

## 4.2 QUICKSORT REVISITÉ

Tout d'abord, nous observons que, dans les modèles de données communément utilisés, les coûts moyens $S_n$ et $K_n$ de *n'importe quel* algorithme sous les modèles de comparaisons de symboles et de comparaisons de clefs, sont reliés par une relation «universelle» $S_n = K_n \cdot O(\log n)$. (Cela provient du fait qu'au plus $O(\log n)$ symboles suffisent, avec une grande probabilité, à distinguer $n$ clefs ; cf. l'analyse de la hauteur des tries dans [6].) Il est plus surprenant d'observer qu'il y a des cas où cette borne est atteinte comme dans le cas de QuickSort ; et d'autres où les coûts restent du même ordre, comme dans QuickSelect.

Nous avons montré que le coût moyen de Quicksort est $O(n \log^2 n)$, et non $O(n \log n)$, lorsque les données sont produites par des sources générales très variées et si *toutes* les opérations élémentaires sont prises en compte. De la même manière, le coût de la recherche dans un arbre binaire de recherche est en moyenne $O(\log^2 n)$, et non $O(\log n)$. Par contraste, le coût de QuickSelect reste $O(n)$, dans les deux cas avec, bien sûr, des constantes différentes. Nos résultats étendent ceux de Fill et Janson [55], Fill et Nakama [56], dont les analyses sont essentiellement dédiées au cas de bits aléatoires uniformes pour les données. Les sources que nous avons considérées incluent les sources sans mémoire («Bernoulli») avec des probabilités indépendantes pour les symboles, les chaînes de Markov, et aussi des sources plus générales ayant une mémoire non bornée.

Une nouvelle idée, dans ce type de problèmes, est d'introduire les *probabilités fondamentales* (la probabilité qu'un mot infini produit par la source admette un certain préfixe). En dehors de cela, l'approche repose sur des méthodes de combinatoire analytique (et particulièrement l'analyse complexe pour l'asymptotique [59]), mais aussi de la théorie de l'information et de la théorie des systèmes dynamiques. Ce travail apparaît donc dans la continuité des analyses que j'avais faites (déjà avec P. Flajolet et B. Vallée) sur les tries [6, 115].

### 4.2.1 *Sources paramétrées*

Avant d'énoncer les résultats obtenus dans [18], on doit fixer quelques définitions et notations. Une *source probabilisée*, produisant des mots infinis de $\Sigma^{\mathbb{N}}$, est définie par la collection $\{p_w\}_{w\in\Sigma^*}$ des *probabilités fondamentales* $p_w$, où $p_w$ est la probabilité qu'un mot infini commence par le préfixe (fini) $w$. On suppose de plus que la suite $\pi_k := \sup\{p_w : w \in \Sigma^k\}$ converge vers 0, lorsque $k \to \infty$.

Pour tout préfixe $w \in \Sigma^*$, on désigne par $p_w^{(-)}$, $p_w^{(+)}$, $p_w$ les probabilités qu'un mot produit par la source commence par un préfixe $w'$ de la même longueur que $w$, et satisfaisant respectivement $w' \prec w$, $w' \succ w$, or $w' = w$. La somme de ces trois probabilités étant égale à 1, cela définit deux réels $a_w, b_w \in [0, 1]$ pour lesquels

$$a_w = p_w^{(-)}, \quad 1 - b_w = p_w^{(+)}, \quad b_w - a_w = p_w.$$

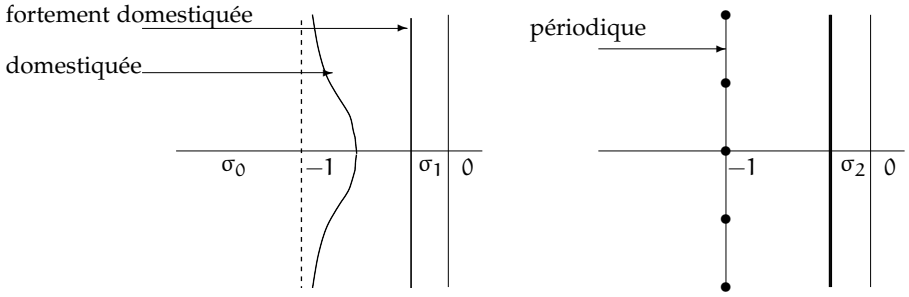



FIGURE 8: *À Gauche* : Les différentes régions sans pôles du plan complexe pour définir les notions de sources faiblement domestiquées, domestiquées. *À Droite* : Région associée à une source périodique.

Étant donné un mot infini $X \in \Sigma^{\mathbb{N}}$, notons $w_k$ son préfixe de longueur $k$. La suite $(a_{w_k})$ est croissante, la suite $(b_{w_k})$ est décroissante, et $b_{w_k} - a_{w_k}$ converge vers 0. Ainsi, il existe un unique réel $p(X) \in [0,1]$ défini comme la limite (commune) de $(a_{w_k})$ et $(b_{w_k})$.

Réciproquement on peut aussi associer tout réel $u$ de l'intervalle $[0,1]$ (excepté peut-être pour un ensemble de réels de mesure nulle correspondants aux extrémités des intervalles fondamentaux[2]) un mot infini $M(u) := (m_1(u), m_2(u), m_3(u), \ldots) \in \Sigma^{\mathbb{N}}$ . Pour tout $k \geqslant 0$, si on définit $w_k$ le préfixe de longueur $k$ tel que $u \in \mathcal{I}_{w_k}$ ($w_k$ est à chaque fois un préfixe de $w_{k+1}$), alors $M(u)$ est la «limite» de cette suite de mots lorsque $k$ tend vers l'infini. Par construction, l'ordre sur les mots de $\Sigma^{\mathbb{N}}$ est compatible avec celui sur l'intervalle $\mathcal{I}$, c'est-à-dire, $M(t) \prec M(u)$ si et seulement si $t < u$. L'intervalle fondamental $\mathcal{I}_w := [a_w, b_w]$ est l'ensemble des réels $u$ pour lesquels $M(u)$ commence par le préfixe $w$. Sa mesure est égale à $p_w$. Nos analyses font intervenir diverses séries de type «Dirichlet» de probabilités fondamentales mais la plus importante reste

$$\Lambda(s) := \sum_{w \in \Sigma^*} p_w^{-s}. \tag{18}$$

La série $\Lambda(s)$ est toujours non définie en s=−1 (puisque $\sum_{w \in \Sigma^k} p_w = 1$). Lorsque $\Lambda(s)$ est analytique dans un domaine contenant un voisinage de $s = -1$, les propriétés de régularité de la source s'expriment grâce celle de $\Lambda$ dans ce domaine. Les théorèmes que nous avons établis considèrent des sources de trois grandes classes que nous avons appelées «*domestiquées*[3]», «*faiblement domestiquées*», et «*périodiques*» (voir la figure **??** pour une vision schématique et [18] pour une définition précise).

SOURCES DYNAMIQUES. Une importante sous-classe est formée par les *sources dynamiques*, étroitement liées aux systèmes dynamiques de l'intervalle [115].

On part d'une partition $\{\mathcal{I}_\sigma\}$ indexée par les symboles $\sigma \in \Sigma$, une application de codage $\tau : \mathcal{I} \to \Sigma$ qui est égale à $\sigma$ sur $\mathcal{I}_\sigma$, une applica-

2. Pour être rigoureux il faudrait définir plus clairement ce qui se passe en cas d'ambiguïté. C'est à peu près le même problème que pour les développements impropres des réels dans un système de numération.

3. En anglais on parle de «tameness».

tion de décalage $T : \mathcal{I} \to \mathcal{I}$ dont la restriction à chaque intervalle $\mathcal{I}_\sigma$ est croissante, inversible, et de classe $\mathcal{C}^2$. Alors le mot $M(u)$ est le mot qui code la trajectoire $(u, Tu, T^2u, \dots)$ à travers l'application $\tau$, $M(u) := (\tau(u), \tau(Tu), \tau(T^2u), \dots)$.

Toutes les sources sans mémoire (Bernoulli) et toutes les sources de type chaîne de Markov appartiennent à ce cadre général : elles correspondent à des applications de décalage qui sont linéaires par morceaux. Par exemple, le système de numération binaire standard est obtenu en considérant $T(x) = \{2x\}$ ($\{\cdot\}$ est la partie fractionnaire). Les sources dynamiques avec une application de décalage non linéaire permettent de prendre en compte tout ou partie du «passé» (par exemple les sources liées aux fractions continues obtenues avec $T(x) = \{1/x\}$). On peut aussi permettre dans ce modèle une densité non uniforme sur $\mathcal{I} = [0, 1]$, ce qui permet de tirer des réels de $[0, 1]$ de manière non uniforme (ce qui assez rare dans les analyses à l'exception notable des travaux de Luc Devroye [49] qui a été le premier à étudier l'effet sur les tries binaires d'une densité non-uniforme).

Dans ce contexte, les propriétés de positivité des opérateurs de Ruelle (généralisés) associés aux sources dynamiques entraînent l'existence d'objets spectraux dominants [115]. En particulier, on prouve l'existence d'une *valeur propre dominante* $\lambda(s)$ définie dans un voisinage de l'axe réel. Cette fonction est omniprésente dans les analyses que nous avons menées (aussi bien dans les tries que dans l'étude de Quicksort et consorts). Les principales caractéristiques intrinsèques de la source, à savoir l'*entropie* $h(S)$ et la *probabilité de coïncidence* $c(S)$, sont indépendantes de la densité initiale $f$ sur l'intervalle unité, et dépendent uniquement du mécanisme de la source à travers les relations

$$h(S) = -\lambda'(1) \qquad c(S) = \lambda(2).$$

Ces opérateurs ne sont pas limités à la modélisation de source mais permettent aussi d'analyser des algorithmes de nature arithmétique (voir par exemple [4] pour une analyse d'un algorithme rapide de calcul du PGCD).

### 4.2.2 *Résultats*

Les algorithmes qui ont été considérés et appliqués à $n$ mots produits indépendamment par la même source $\mathcal{S}$ sont : l'algorithme de tri QuickSort, l'algorithme de recherche QuickSelect pour le minimum et le maximum, notés QuickSelMin ou QuickSelMax, et finalement QuickSelRand, qui correspond à QuickSelect$(m, n)$, où le rang $m$ de la clef recherchée est lui-même choisi aléatoirement et uniformément dans $[1 \,..\, n]$ ; voir, par exemple, le livre de B. Sedgewick [109] pour une description de ces algorithmes standard. Les résultats obtenus sont les suivants.

**Théorème** *(i) Pour une source domestiquée $\mathcal{S}$, le nombre moyen $\nu_n$ de comparaisons de symboles de QuickSort$(n)$ est*

$$\nu_n = \frac{1}{h(\mathcal{S})} n \log^2 n + a(\mathcal{S}) n \log n + b(\mathcal{S}) n + o(n), \qquad (19)$$

*où* $h(\mathcal{S}) := \lim_{k\to\infty} \left[ -\frac{1}{k} \sum_{w\in\Sigma^k} p_w \log p_w \right]$ *est l'entropie de la source.*

*(ii) Pour une source périodique, un terme $nP(\log n)$ doit être ajouté à la valeur de (19), où $P(u)$ est une fonction continue et périodique, donnée par sa série de Fourier.*

**Théorème**
*Pour une source faiblement domestiquée $\mathcal{S}$, le nombre moyen de comparaisons $\mu_n^{(-)}$ pour QuickSelMin$(n)$ et $\mu_n^{(+)}$ pour QuickSelMax$(n)$, satisfont pour un certain $\delta < 1$, et $\varepsilon = \pm$,*

$$\mu_n^{(\varepsilon)} = \alpha_{\mathcal{S}}^{(\varepsilon)} n + O(n^\delta), \qquad \alpha_{\mathcal{S}}^{(\varepsilon)} := 2 \sum_{w\in\Sigma^*} p_w \left[ 1 - \frac{p_w^{(\varepsilon)}}{p_w} \log\left( 1 + \frac{p_w}{p_w^{(\varepsilon)}} \right) \right].$$

**Théorème**
*Pour une source faiblement domestiquée $\mathcal{S}$, le nombre moyen $\mu_n$ de comparaisons de l'algorithme QuickSelRand$(n)$ satisfait, pour un certain $\delta < 1$,*

$$\mu_n = \gamma_{\mathcal{S}}\, n + O(n^\delta),$$

*avec*

$$\gamma_{\mathcal{S}} := \sum_{w\in\Sigma^*} p_w^2 \left[ 2 + \frac{1}{p_w} + \sum_{\varepsilon=\pm} \left[ \log\left( 1 + \frac{p_w^{(\varepsilon)}}{p_w} \right) - \left( \frac{p_w^{(\varepsilon)}}{p_w} \right)^2 \log\left( 1 + \frac{p_w}{p_w^{(\varepsilon)}} \right) \right] \right].$$

*Les constantes $\alpha_{\mathcal{S}}^{(\varepsilon)}$ et $\gamma_{\mathcal{S}}$ dépendent de la source $\mathcal{S}$. Les conditions de l'application du théorème sont discutées dans [18].*

Les expressions précédentes se calculent agréablement dans le cas, classique, de la source binaire $\mathcal{B}$, où les clefs sont des mots binaires infinis dont les symboles sont tirés dans $\{0, 1\}$ aléatoirement et indépendamment avec probabilité $1/2$. Alors les constantes ont les formes suivantes. On a : $h(\mathcal{B}) = \log 2$, et

$$\alpha_{\mathcal{B}}^{(\varepsilon)} = 4 + 2 \sum_{\ell\geqslant 0} \frac{1}{2^\ell} \sum_{k=1}^{2^\ell - 1} \left[ 1 - k \log\left( 1 + \frac{1}{k} \right) \right]$$

$$\gamma_{\mathcal{B}} = \frac{14}{3} + 2 \sum_{\ell=0}^{\infty} \frac{1}{2^{2\ell}} \sum_{k=1}^{2^\ell - 1} \left[ k + 1 + \log(k+1) - k^2 \log\left( 1 + \frac{1}{k} \right) \right].$$

Cela permet de calculer avec une bonne précision numérique ces constantes. Ainsi on a $\alpha_{\mathcal{B}}^{(\varepsilon)} = 5.27937\,82410\,80958\cdots$. (Ces expressions simplifient et étendent celles obtenues par Fill–Nakama [56] et Grabner–Prodinger [70].)

### 4.2.3 *Stratégie générale.*

L'étude est faite sous un modèle de source général, paramétré par l'intervalle $\mathcal{I} = [0, 1]$. On peut décomposer cette analyse en trois étapes. Les deux premières sont de nature essentiellement *algébrique*, tandis que la dernière repose sur de l'*analyse complexe*.

**Étape** $(a)$**.** Nous montrons d'abord que le nombre moyen $S_n$ de comparaisons de symboles opérées par un algorithme $\mathcal{A}(n)$ appliqué à $n$ mots de la source $\mathcal{S}$ peut s'exprimer à l'aide de deux objets : (i) la *densité* $\phi_n$ de l'algorithme, qui dépend uniquement de l'algorithme et donne une mesure du nombre moyen de comparaisons pour une paire de *clefs* (vues comme des points de $\mathcal{I} = [0, 1]$) ; (ii) la famille de *triangles fondamentaux*, qui dépendent seulement de la source et décrivent les régions correspondant à des paires de mots avec le même préfixe.

La *densité* pour un algorithme $\mathcal{A}(n)$ opérant sur un ensemble de $n$ mots produits indépendamment par la même source probabiliste $\mathcal{S}$ (et correspondant à autant de réels tirés uniformément et indépendamment sur l'intervalle $\mathcal{I} = [0, 1]$) est définie comme suit

$$\begin{aligned}\phi_n(u, t)\, du\, dt := \text{« le nombre moyen de comparaisons entre} \\ \text{deux clefs de paramètres } u' \in [u, u + du] \\ \text{et } t' \in [t, t + dt] \text{ effectuées par } \mathcal{A}(n)\text{»}.\end{aligned}$$

La moyenne considérée considère tous les ensembles de $n$ points sur $[0, 1]$ et tous les ordres possibles sur ces points en supposant que les deux clefs $u'$ et $t'$ sont dans cette ensemble. À ce stade, si l'on souhaite simplement calculer le nombre moyen de comparaisons de *clefs* (cadre classique) effectuées par $\mathcal{A}(n)$, on doit évaluer

$$\int_{\mathcal{T}} \phi_n(u, t)\, du\, dt.$$

Pour compter le nombre de comparaisons de *symboles*, on doit considérer pour chaque préfixe $w \in \Sigma^*$, le *triangle fondamental* de $w$, noté $\mathcal{T}_w$, est le triangle construit sur l'intervalle fondamental $\mathcal{I}_w := [a_w, b_w]$ de $w$ ; i.e., $\mathcal{T}_w := \{(u, t) : a_w \leqslant u \leqslant t \leqslant b_w\}$ et qui correspond à toutes les paires de mots qui partagent au moins $w$ comme un préfixe commun. On définit $\mathcal{T} \equiv \mathcal{T}_\varepsilon := \{(u, t) : 0 \leqslant u \leqslant t \leqslant 1\}$.

On relie ensuite le nombre $S_n$ de comparaisons de symboles à la densité de l'algorithme $\phi_n$ en passant par les triangles $\mathcal{T}_w$. On définit d'abord pour une fonction $g$ (et pourvu que cela converge) la transformation intégrale

$$\mathcal{J}[g] := \sum_{w \in \Sigma^*} \int_{\mathcal{T}_w} g(u, t)\, du\, dt.$$

On a alors

$$S_n = \mathcal{J}[\phi_n] = \sum_{w \in \Sigma^*} \int_{\mathcal{T}_w} \phi_n(u, t)\, du\, dt. \tag{20}$$

Cette formule est établie en faisant intervenir la *fonction de coïncidence* $\gamma(u,t)$ définie comme la longueur du plus long préfixe commun entre $M(u)$ et $M(t)$, i.e., $\gamma(u,t) := \max\{\ell : m_j(u) = m_j(t), \forall j \leqslant \ell\}$. Alors le nombre de comparaisons entre symboles nécessaires pour comparer $M(u)$ et $M(t)$ est $\gamma(u,t)+1$ et le nombre moyen de comparaisons s'écrit

$$S_n = \int_{\mathcal{T}} [\gamma(u,t)+1]\, \phi_n(u,t)\, du\, dt.$$

On obtient finalement l'expression (20) en utilisant deux identités

$$\sum_{\ell \geqslant 0} (\ell+1)\mathbf{1}_{[\gamma=\ell]} = \sum_{\ell\geqslant 0} \mathbf{1}_{[\gamma\geqslant\ell]} \qquad [\gamma \geqslant \ell] = \bigcup_{w\in\Sigma^\ell} (\mathcal{I}_w \times \mathcal{I}_w)\,.$$

La première égalité est valable pour toute fonction à valeurs entières $\gamma$ ($\mathbf{1}_A$ est la fonction indicatrice de $A$). La seconde vient de la définition de $\gamma$ et des intervalles fondamentaux $\mathcal{I}_w$. Enfin, le domaine $\mathcal{T} \cap [\gamma \geqslant \ell]$ est une union de triangles fondamentaux.

**Étape** (b)**.** Cette étape consiste à calculer la densité de l'algorithme. Partant des formules de bases dues à Fill et Janson [55], nous introduisons le *modèle de Poisson*, où le nombre de clefs n'est plus fixé mais est lui-même une variable aléatoire obéissant à une loi de Poisson de paramètre $Z$

$$\mathbf{P}[N = n] = e^{-Z}\frac{Z^n}{n!}.$$

Ce modèle est particulièrement adapté à nos besoins. En effet La valeur moyenne $\tilde{S}_Z$ du nombre de comparaisons dans ce modèle s'exprime

$$\tilde{S}_Z := e^{-Z}\sum_{n\geqslant 0}\frac{Z^n}{n!}S_n = e^{-Z}\sum_{n\geqslant 0}\frac{Z^n}{n!}\mathcal{J}[\phi_n] = \mathcal{J}\left[e^{-Z}\sum_{n\geqslant 0}\frac{Z^n}{n!}\phi_n\right] = \mathcal{J}[\tilde{\phi}_Z],$$

et fait intervenir la densité de l'algorithme mais *poissonisée* $\tilde{\phi}_Z$, définie par

$$\tilde{\phi}_Z(u,t) := e^{-Z}\sum_{n\geqslant 0}\frac{Z^n}{n!}\phi_n(u,t).$$

Nous utilisons les principes généraux des calculs décrits dans [56] et obtenons une expression de la densité «poissonisée» pour chaque algorithme. Par exemple, la densité poissonisée de l'algorithme QuickSort admet l'expression

$$\tilde{\phi}_Z(u,t) = 2(t-u)^{-2}\, f_1(Z(t-u)), \qquad \text{avec } f_1(\theta) := e^{-\theta} - 1 + \theta.$$

Le nombre moyen de comparaisons de QuickSort dans le modèle de Poisson vérifie donc

$$\tilde{S}_Z = 2\mathcal{J}[(t-u)^{-2}\cdot f_1(Z(t-u))].$$

En développant la fonction $f_1$ et en échangeant l'ordre de l'intégrale et de la sommation (sous conditions évidemment), on obtient une expression sous forme d'une somme alternée

$$\tilde{S}_Z = \sum_{k=2}^{\infty}(-1)^k\varpi(-k)\frac{Z^k}{k!}, \tag{21}$$

où $\varpi(s)$ est une série de type Dirichlet définie pour $\Re s \leqslant -2$ par

$$\varpi(s) = 2\mathcal{J}[(t-u)^{-(2+s)}] = \frac{\Lambda(s)}{s(s+1)}, \tag{22}$$

la dernière égalité provenant juste du calcul de l'intégrale[4]. En prenant en compte que $S_n$ est relié à $\tilde{S}_Z$ via la relation $S_n = n![Z^n](e^Z \tilde{S}_Z)$, on trouve l'expression (exacte) suivante pour $S_n$

$$S_n = \sum_{k=2}^{n} \binom{n}{k} (-1)^k \varpi(-k), \qquad \text{for } n \geqslant 2. \tag{23}$$

Cette expression est certes exacte, mais il convient d'en étudier l'asymptotique pour en déduire quelque chose de plus informatif.

**Étape** (c)**.** Les formules exactes obtenues pour les espérances mettent en jeu deux types de quantités : le cardinal $n$ de l'ensemble de clefs à traiter (qui tend vers l'infini) et les intervalles fondamentaux qui résident dans l'intervalle $\mathcal{I}$ (et qui ont une mesure tendant vers 0). Nous menons l'analyse asymptotique en considérant des *représentations intégrales complexes* de type Nörlund–Rice. Pour chaque paire algorithme–source, une série de Dirichlet rend compte à la fois des propriétés de la source et des caractéristiques de l'algorithme — Il s'agit d'une *série de Dirichlet modifiée*, dénotée par $\varpi(s)$, dont la structure des *singularités dans le plan complexe* conditionne les estimations asymptotiques finalement obtenues.

On a recours à une représentation intégrale pour mieux cerner le comportement asymptotique. En effet la suite $\varpi(-k)$ se prolonge en une fonction analytique $\varpi(s)$ et, sous conditions d'analycité et de régularité de $\varpi(s)$, on a l'expression suivante appelée *représentation de Nörlund–Rice* pour $n$ assez grand et un réel $-2 < d < -1$ bien choisi

$$S_n = \frac{1}{2i\pi} \int_{d-i\infty}^{d+i\infty} \varpi(s) \frac{n!}{s(s+1)\cdots(s+n)}\, ds. \tag{24}$$

De manière assez standard, il suffit maintenant d'étudier ce qui se passe à la singularité dominante (ici $s = -1$) et à effectuer un calcul de résidus. Plus précisément, dans le cas de QuickSort, on doit étudier la fonction

$$\frac{\varpi(s)}{s+1} = \frac{2}{s+1}\mathcal{J}[(t-u)^{-(2+s)}] = 2\frac{\Lambda(s)}{s(s+1)^2}\,. \tag{25}$$

Selon le type de source, on peut avoir différents comportements. Une première classification des sources est proposée dans [18] et a été

4. la situation est vraiment analogue à celle des tries puisque, par exemple, l'expression de la longueur de cheminement moyenne $L_Z$ (le «coût de construction») du trie dans le modèle de Poisson de paramètre $Z$ est d'après [6]

$$L_Z = \sum_{w \in \Sigma^*} Z p_w (1 - e^{-Z p_w}).$$

Si cette expression a été analysée grâce à la transformée de Mellin dans [6], on peut aussi utiliser la méthodologie présentée ici. En développant en série entière $Z(1-e^{-Z})$, on obtient les mêmes expressions que (21) et (23) mais en posant $\varpi(s) = -s\Lambda(s)$.

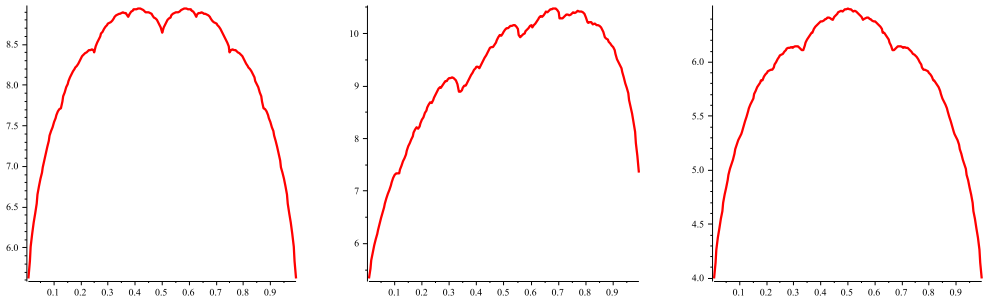


FIGURE 9: Graphes des fonctions $\rho_S(\alpha)$ pour $\alpha \in [0,1]$ et les trois sources : Bern$(\frac{1}{2},\frac{1}{2})$, Bern$(\frac{1}{3},\frac{2}{3})$, and Bern$(\frac{1}{3},\frac{1}{3},\frac{1}{3})$. Ces courbes illustrent le caractère fractal des constantes apparaissant dans $\mathsf{QuickQuant}_\alpha$.

grandement raffinée depuis dans [58, 105] : cette classification cherche à relier les propriétés arithmétiques ou géométriques des sources aux propriétés analytiques de la fonction $\Lambda(s)$ correspondante.

Ainsi si la source est «domestiquée» (selon la terminologie), on observe un pôle triple en $s = -1$ : un pôle d'ordre 1 provenant de $\Lambda(s)$ et un pôle d'ordre 2 provenant du facteur $1/(s+1)^2$ ce qui permet de déduire directement [5] un comportement en $O(n \log^2 n)$.

La source peut également être «périodique» (toujours selon la même terminologie), amenant d'autres pôles (simples) sur la droite $\Re s = -1$ et entraînant un terme périodique de la forme $nP(\log n)$, où $P$ est une fonction périodique décrite sous forme d'une série de Fourier.

Il est également possible d'exhiber des sources pour lesquelles le comportement va être un peu différent, par exemple des sources dynamiques intermittentes [116]. En rajoutant un pôle, on va multiplier la complexité par un facteur $\log n$.

Dans [18], nous avons amorcé l'étude des constantes apparaissant dans l'asymptotique de $\mathsf{QuickQuant}_\alpha$, et même pour des sources simples (voir la figure 8), on observe des comportements assez étranges (de type «brocoli» dans la terminologie de P. Flajolet) qu'il conviendrait de mieux expliquer (position des maxima par exemple).

## 4.3 CONCLUSION

Dans ce chapitre, nous avons introduit une autre manière plus réaliste, d'aborder l'analyse d'algorithmes classiques de tri. En collaboration avec H. Nguyen Thi et B. Vallée, nous souhaitons appliquer cette méthodologie à d'autres algorithmes de tri et de recherche. Même si en général, le résultat attendu est d'obtenir une complexité multipliée par un facteur $\log n$, on a vu que ce n'est pas toujours le cas (exemple de QuickSelect). Notre approche permet aussi de calculer les constantes assez simplement pour pouvoir mieux comparer les algorithmes entre eux.

5. Sur les tries (voir la note de bas de page précédente) on a bien un pôle en $s = -1$ d'ordre 2 si la source est «domestiquée» donnant une longueur moyenne de cheminement externe en $O(n \log n)$.

# 5

# CONCLUSION

Dans ce mémoire, j'ai examiné différents aspects d'un objet simple mais omniprésent en informatique : la séquence de symboles (appelée selon le contexte mot ou chaîne de caractères). La notion de mot est au carrefour de domaines comme la théorie de l'information et la théorie des langages. S'il est simple, il reste fondamental : nous n'avons, au plus bas niveau, que cela à disposition [1] puisqu'il arrive toujours un moment où une donnée doit être encodée en symboles stockables en mémoire.

La quantité d'information croissante de données mise à disposition et qu'on peut stocker, par exemple des génomes d'individus ou des documents numérisés, justifie que les algorithmes et les structures de données qui les manipulent soient optimisés. En conséquence, les besoins d'analyse se font sentir pour guider le choix et la conception des programmes qui manipulent ces données. L'analyse en moyenne est ici particulièrement adaptée puisque les données atteignent une variété et des volumes tellement importants que c'est le cas typique qui traduit le mieux la complexité et non pas le cas le pire. Cela évidemment pose le problème de la modélisation de données qui reste encore très épineux, puisque l'on souhaite deux choses contradictoires : un modèle au plus prêt des données, qui traduise vraiment leurs spécificités, mais aussi un modèle permettant de donner des résultats, c'est-à-dire de prédire les performances (et on comprend vite que le modèle doit donc rester relativement simple pour qu'il subsiste un espoir de le traiter !). J'ajoute que la modélisation est également la pierre de base sur laquelle s'appuient les algorithmes de compression. De plus il existe de multiples manières d'envisager les mots ou leur structure. Par exemple on peut considérer les fonctions booléennes comme des mots, ou encore, tenter de mieux comprendre les chevauchements d'un mot ou à l'intérieur d'un ensemble de mots.

Je pense qu'il y a là encore grandement matière à la réflexion et à de nouveaux problèmes intéressants pour l'avenir.

1. En attendant les ordinateurs quantiques...

# RÉFÉRENCES BIBLIOGRAPHIQUES

La liste de mes publications se trouve page vii. Je liste ci-après les références bibliographiques citées dans ce mémoire.


[21] Abrahams, J. Code and parse trees for lossless source encoding. *Commun. Inf. Syst. 1*, 2 (2001), 113–146.

[22] Aho, A., and Corasick, M. Efficient String Matching. *CACM 18*, 6 (1975), 333–340.

[23] Apostolico, A., and Giancarlo, R. The Boyer-Moore-Galil string searching strategies revisited. *SIAM J. Comput. 15*, 1 (1986), 98–105.

[24] Baer, M. B. *Coding for General Penalties*. PhD thesis, Stanford University, 2003.

[25] Baeza-Yates, R., Choffrut, C., and Gonnet, G. On Boyer-Moore automata. *Algorithmica 12*, 4/5 (1994), 268–292.

[26] Bannai, H., Inenaga, S., Shinohara, A., and Takeda, M. Inferring strings from graphs and arrays. In *Mathematical Foundations of Computer Science* (2003), B. Rovan and P. Vojtás, Eds., vol. 2747 of *Lecture Notes in Computer Science*, Springer, pp. 208–217.

[27] Bassino, F., Béal, M.-P., and Perrin, D. A finite state version of the Kraft-McMillan theorem. *SIAM Journal on Computing 30*, 4 (2000), 1211–1230.

[28] Bassino, F., Giambruno, L., and Nicaud, C. The average state complexity of rational operations on finite languages. *Int. J. Found. Comput. Sci. 21*, 4 (2010), 495–516.

[29] Bedford, T., Keane, M., and Series, C. *Ergodic Theory, Symbolic Dynamics and Hyperbolic Spaces*. Oxford University Press, 1991.

[30] Bender, E. A., and Kochman, F. The distribution of subword counts is usually normal. *European Journal of Combinatorics 14* (1993), 265–275.

[31] Bentley, J., and Sedgewick, R. Fast algorithms for sorting and searching strings. In *Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (January 1997), SIAM Press, pp. 360–369. New Orleans.

[32] Berstel, J., and Boasson, L. The set of Lyndon words is not context-free. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 63* (1997), 139–140.

[33] Berstel, J., and Perrin, D. *Theory of codes*. Academic Press, 1985.

[34] Berstel, J., and Perrin, D. Algorithms on words. In *Applied combinatorics on words*, M. Lothaire, Ed., vol. 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005, ch. 1, pp. 1–105.

[35] Berstel, J., and Pocchiola, M. Average cost of Duval's algorithm for generating Lyndon words. *Theoretical Computer Science 132*, 1-2 (1994), 415–425.

[36] Bourdon, J., and Vallée, B. Generalized pattern matching statistics. In *Proc. Colloquium on Mathematics and Computer Science : Algorithms, Trees, Combinatorics and Probabilities* (2002), Birkhauser, Trends in Mathematics, pp. 249–265.

[37] Bourdon, J., and Vallée, B. Pattern matching statistics on correlated sources. In *Proc. of LATIN'06* (2006), vol. 3887 of *LNCS*, pp. 224–237.

[38] Boyer, R. S., and Moore, J. S. A fast string searching algorithm. *Commun. ACM 20*, 10 (1977), 762–772.

[39] Broutin, N., and Devroye, L. An analysis of the height of tries with random weights on the edges. *Combinatorics, Probability & Computing 17*, 2 (2008), 161–202.

[40] Chassaing, P., and Zohoorian Azad, E. Asymptotic behavior of some factorizations of random words. (hal-00475379).

[41] Chen, K., Fox, R., and Lyndon, R. Free differential calculus IV : The quotient groups of the lower central series. *Ann. Math. 58* (1958), 81–95.

[42] Chomsky, N., and Schützenberger, M. P. The algebraic theory of context-free languages. *Computer Programming and Formal Languages,* (1963), 118–161. P. Braffort and D. Hirschberg, eds, North Holland.

[43] Clampett, H. A. Randomized binary searching with tree structures. *Comm. ACM 7*, 3 (Mar. 1964), 163–165.

[44] Crochemore, M., Hancart, C., and Lecroq, T. *Algorithms on Strings*. Cambridge University Press, Cambridge, UK, 2007. 392 pages.

[45] Crochemore, M., and Ilie, L. Computing Longest Previous Factor in linear time and applications. *Inf. Process. Lett. 106*, 2 (2008), 75–80.

[46] Crochemore, M., Ilie, L., and Smyth, W. F. A simple algorithm for computing the Lempel-Ziv factorization. In *18th Data Compression Conference* (2008), J. A. Storer and M. W. Marcellin, Eds., IEEE Computer Society, Los Alamitos, CA, pp. 482–488.

[47] Crochemore, M., and Lecroq, T. Tight bounds on the complexity of the Apostolico-Giancarlo algorithm. *Information Processing Letters 63*, 4 (1997), 195–203.

[48] Crochemore, M., and Rytter, W. *Jewels of Stringology*. World Scientific Publishing, Hong-Kong, 2002. 310 pages.

[49] Devroye, L. A probabilistic analysis of the height of tries and of the complexity of triesort. *Acta Informatica 21* (1984), 229–237.

[50] Devroye, L. *Probabilistic Methods for Algorithmic Discrete Mathematics*. Springer, 1998, ch. Branching Processes and Their Applications in the Analysis of Tree Structures and Tree Algorithms.

[51] Duval, J.-P. Factorizing words over an ordered alphabet. *Journal of Algorithms 4* (1983), 363–381.

[52] Duval, J.-P., Lecroq, T., and Lefebvre, A. Border array on bounded alphabet. *Journal of Automata, Languages and Combinatorics 10*, 1 (2005), 51–60.

[53] Duval, J.-P., Lecroq, T., and Lefebvre, A. Efficient validation and construction of border arrays. In *Proceedings of 11th Mons Days of Theoretical Computer Science* (Rennes, France, 2006), pp. 179–189.

[54] Feller, W. *An Introduction to Probability Theory and Its Applications*, vol. 2. John Wiley, 1971.

[55] Fill, J. A., and Janson, S. The number of bit comparisons used by Quicksort : An average-case analysis. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04)* (2001), pp. 293–300.

[56] Fill, J. A., and Nakama, T. Analysis of the expected number of bit comparisons required by quickselect. *Algorithmica 58*, 3 (2010), 730–769.

[57] Flajolet, P., Gourdon, X., and Panario, D. The complete analysis of a polynomial factorization algorithm over finite fields. *J. Algorithms 40*, 1 (2001), 37–81.

[58] Flajolet, P., Roux, M., and Vallée, B. Digital trees and memoryless sources : from arithmetics to analysis. *DMTCS Proceedings 0*, 01 (2010).

[59] Flajolet, P., and Sedgewick, R. *Analytic Combinatorics*. Cambridge University Press, 2009.

[60] Franek, F., Gao, S., Lu, W., Ryan, P. J., Smyth, W. F., Sun, Y., and Yang, L. Verifying a Border array in linear time. *J. Combinatorial Math. and Combinatorial Computing 42* (2002), 223–236.

[61] Fredricksen, H., and Maiorana, J. Necklaces of beads in k colors and k-ary de Bruijn sequences. *Discrete Math. 23*, 3 (1978), 207–210.

[62] Gallager, R. G., and Voorhis, D. C. V. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory* (1975), 228–230.

[63] Golin, M. J., and Ma, K. K. Algorithms for constructing infinite huffman codes. Technical Report HKUST-TCSC-2004-07, HKUST, Hong Kong, China, July 2004.

[64] Golomb, S. Irreducible polynomials, synchronizing codes, primitive necklaces and cyclotomic algebra. In *Proc. Conf Combinatorial Math. and Its Appl.* (Chapel Hill, 1969), Univ. of North Carolina Press, pp. 358–370.

[65] Golomb, S. W. Run length encodings. *IEEE Transactions on Information Theory IT-12*, 399-401 (1966).

[66] Golomb, S. W. Sources which maximize the choice of a Huffman coding tree. *Information and Control 45* (jun 1980), 263–272.

[67] Gonnet, G., and Baeza-Yates, R. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1991.

[68] Goulden, I., and Jackson, D. An inversion theorem for clusters decompositions of sequences with distinguished subsequences. *J. London Math. Soc. 2*, 20 (1979), 567–576.

[69] Goulden, I. P., and Jackson, D. M. *Combinatorial enumeration*. John Wiley & Sons, 1983.

[70] Grabner, P. J., and Prodinger, H. On a constant arising in the analysis of bit comparisons in quickselect. *Quaest. Math. 31*, 4 (2008), 303–306.

[71] Guibas, L., and Odlyzko, A. Periods in strings. *J. Combin. Theory A*, 30 (1981), 19–42.

[72] Guibas, L., and Odlyzko, A. Strings overlaps, pattern matching, and non-transitive games. *J. Combin. Theory A*, 30 (1981), 108–203.

[73] Gusfield, D. *Algorithms on strings, trees and sequences : computer science and computational biology*. Cambridge University Press, Cambridge, UK, 1997. 534 pages.

[74] Huffman, D. A. A method for the construction of minimum-redundancy codes. In *Proceedings of the IRE* (1952), vol. 40, pp. 1098–1101.

[75] I, T., Inenaga, S., Bannai, H., and Takeda, M. Verifying and enumerating parameterized border arrays. *Theoretical Computer Science*, 0 (2011), –.

[76] Jacquet, P., and Szpankowski, W. Autocorrelation on words and its applications - analysis of suffix trees by string-ruler approach. *J. Combin.Theory Ser. A 66* (1994), 237–269.

[77] Jacquet, P., and Szpankowski, W. Analytic approach to pattern matching. In *Applied combinatorics on words*, M. Lothaire, Ed., vol. 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005, ch. 7, pp. 329–398.

[78] Kato, A., Han, T. S., and Nagaoka, H. Huffman coding with an infinite alphabet. *IEEE Trans. Inf. Theory 42* (1996), 977–984.

[79] Knuth, D. E. The average time for carry propagation. *Indagationes Mathematicae 40* (1978), 238–242.

[80] Knuth, D. E. *The Art of Computer Programming*, third ed., vol. 3 : Sorting and Searching. Addison-Wesley, 1998.

[81] Knuth, D. E., Jr, J. H. M., and Pratt, V. R. Fast pattern matching in strings. *SIAM J. Comput. 6*, 1 (1977), 323–350.

[82] Kong, Y. Extension of Goulden-Jackson cluster method on pattern occurrences in random sequences and comparison with Régnier Szpankowski method. *J. of Difference Equations and Applications 11*, 15 (2005), 1265–1271.

[83] Linder, T., Tarokh, V., and Zeger, K. Existence of optimal prefix codes for infinite source alphabets. *IEEE Trans. Inf. Theory 43* (1997), 2026–2028.

[84] Lothaire. *Combinatorics on Words*. Addison-Wesley, Reading, Mass., 1983.

[85] Lothaire, M. *Applied combinatorics on words*, vol. 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2005. With a preface by Jean Berstel and Dominique Perrin.

[86] Mahmoud, H. *Evolution of Random Search Trees*. John Wiley, New York, 1992.

[87] MARCHAND, R., AND AZAD, E. Z. Limit law of the length of the standard right factor of a Lyndon word. *Combinatorics, Probability & Computing 16*, 3 (2007), 417–434.

[88] MARTÍNEZ, C., AND ROURA, S. Randomized binary search trees. *J. ACM 45*, 2 (1998), 288–323.

[89] MERHAV, N., SEROUSSI, G., AND WEINBERGER, M. J. Optimal prefix codes for sources with two-sided geometric distributions. *IEEE Transactions on Information Theory 46*, 1 (2000), 229–236.

[90] NICODÈME, P. Regexpcount, a symbolic package for counting problems on regular expressions and words. *Fundamenta Informaticae 56*, 1-2 (2003), 71–88.

[91] NICODÈME, P., SALVY, B., AND FLAJOLET, P. Motif statistics. In *Algorithms, ESA'99* (1999), J. Nešetril, Ed., vol. 1643 of *Lecture Notes in Computer Science*, pp. 194–211.

[92] NICODÈME, P., SALVY, B., AND FLAJOLET, P. Motif statistics. *Theoretical Computer Science 287*, 2 (2002), 593–618.

[93] PANARIO, D., AND RICHMOND, B. Smallest components in decomposable structures : exp-log class. *Algorithmica 29* (2001), 205–226.

[94] PRUM, B., RODOLPHE, F., AND DE TURCKHEIM, E. Finding words with unexpected frequencies in deoxyribonucleic acid sequences. *J. R. Statist. Soc. B 57*, 1 (1995), 205–220.

[95] RÉGNIER, M. A unified approach to word occurrences probabilities. *Discrete Applied Mathematics 104*, 1 (2000), 259–280. Special issue on Computational Biology.

[96] RÉGNIER, M., AND SZPANKOWSKI, W. On the approximate pattern occurrences in a text. In *SEQUENCES '97 : Proceedings of the Compression and Complexity of Sequences 1997* (Washington, DC, USA, 1997), IEEE Computer Society, p. 253.

[97] RÉGNIER, M., AND SZPANKOWSKI, W. On pattern frequency occurrences in a markovian sequence ? *Algorithmica 22*, 4 (1998), 631–649. This paper was presented in part at the 1997 International Symposium on Information Theory, Ulm, Germany.

[98] REINERT, G., AND SCHBATH. Compound Poisson and Poisson process approximations for occurrences of multiple words in Markov chains. *J. Comp. Biol. 5* (1998), 223–253.

[99] REINERT, G., SCHBATH, S., AND WATERMAN, M. Probabilistic and statistical properties of words : an overview. *J. Comp. Biol. 7* (2000), 1–46.

[100] REINERT, G., SCHBATH, S., AND WATERMAN, M. S. Statistics on words with applications to biological sequences. In *Applied combinatorics on words*, M. Lothaire, Ed., vol. 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005, ch. 6, pp. 251–328.

[101] REUTENAUER, C. *Free Lie algebras*. Oxford University Press, 1993.

[102] RICE, R. F. Some practical universal noiseless coding techniques. Tech. Rep. JPL-79-22, JPL, Pasadena, CA, 1979.

[103] RIVEST, R. L. Partial match retrieval algorithms. *SIAM J. Comput. 5* (1976), 19–50.

[104] Roquain, E., and Schbath, S. Improved compound poisson approximation for the number of occurrences of multiple words in a stationary markov chain. *Adv. Appl. Prob.*, 39 (2007), 1–13.

[105] Roux, M. *Séries de Dirichet et analyse en moyenne d'algorithmes de réduction des réseaux*. PhD thesis, Université de Caen, 2011.

[106] Ruelle, D. *Dynamical Zeta Functions for Piecewise Monotone Maps of the Interval*, vol. 4 of *CRM Monograph Series*. American Mathematical Society, Providence, 1994.

[107] Schbath, S. Compound Poisson approximation of word counts in DNA sequences. *ESAIM Probab. Statist. 1* (1995), 1–16.

[108] Sedgewick, R. *Quicksort*. Garland Pub. Co., New York, 1980. Reprint of Ph.D. thesis, Stanford University, 1975.

[109] Sedgewick, R. *Algorithms in C : Fundamentals, Data Structures, Sorting, Searching*, third ed. Addison–Wesley, Reading, Mass., 1988.

[110] Sedgewick, R. *Algorithms in C, Parts 1–4*, third ed. Addison–Wesley, Reading, Mass., 1998.

[111] Sedgewick, R., and Flajolet, P. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Publishing Company, 1996.

[112] Szpankowski, W. *Average Case Analysis of Algorithms on Sequences*. Series in Discrete Mathematics and Optimization. John Wiley & Sons, 2001.

[113] Tompa, M., Li, N., Bailey, T., Church, G., De Moor, B., Eskin, E., Favorov, A., Frith, M., Fu, Y., Kent, J., Makeev, V., Mironov, A., Noble, W., Pavesi, G., Pesole, G., Régnier, M., Simonis, N., Sinha, S., Thijs, G., van Helden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C., and Zhu, Z. An assessment of computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology 23*, 1 (January 2005), 137 – 144.

[114] Trabb Pardo, L. Set representation and set intersection. Tech. rep., Stanford University, 1978.

[115] Vallée, B. Dynamical sources in information theory : Fundamental intervals and word prefixes. *Algorithmica 29*, 1 (2001), 262–306.

[116] Vallée, B. Euclidean dynamics. *Discrete and Continuous Dynamical Systems 1*, 15 (2006), 281–352.

[117] Weinberger, M. J., Seroussi, G., and Sapiro, G. The LOCO-I lossless image compression algorithm : Principles and standardization into JPEG-LS. *IEEE Trans. Image Proc. 9*, 8 (Aug. 2000), 1309–1324.