Exemples d'analyse d'algorithmes: calculs de PGCD et algorithmes de tri et de recherche

Julien Clément, Loïck Lhote

Chapitre 1

Exemples d'analyses d'algorithmes : calculs de PGCD et algorithmes de tri et de recherche

Julien Clément
Loïck Lhote

Dans ce cours, nous allons commencer par définir le cadre de l'analyse d'algorithmes tel qu'il a été fondé par Don Knuth (avec sa série de volumes The Art of Computer Programming*) dans les années 60, et popularisé et développé depuis notamment par Philippe Flajolet [25, 44] en France. Nous utilisons essentiellement des techniques issus de la combinatoire analytique. Nous ne prétendons pas couvrir tout le champ de l'analyse d'algorithmes, puisqu'il existe d'autres approches comme celles basées sur des techniques probabilistes.*

Nous examinons dans ce cours deux types de problèmes qui correspondent à des travaux récents [3, 10]. Nous étudions en premier lieu l'analyse en moyenne et en distribution du calcul de PGCD *de plusieurs polynômes ou entiers. Le cas des polynômes est traité via une approche relativement standard en combinatoire analytique à base de manipulations formelles de séries génératrices. Le cas des entiers est évidemment plus difficile et technique mais, à haut niveau, les étapes de l'analyse sont similaires. Ensuite nous changeons de champ d'application en présentant une méthode générale appliquée aux algorithmes classiques de tri et de recherche. Cette méthode cherche à analyser*

plus finement ces algorithmes en tenant compte du mécanisme (qu'on appellera source) qui produit les données en entrée.

1.1 Analyse d'algorithmes

1.1.1 Contexte

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 :

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

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. *La problématique générale de ce chapitre est celui de l'analyse de complexité en moyenne*.

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 à choisir une permutation de manière uniforme dans $\{1, \ldots, n\}$.

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).

1.1.2 Exemples d'algorithmes

Dans ce chapitre, nous illustrons les concepts de l'analyse en moyenne sur plusieurs exemples d'algorithmes. Le premier est un algorithme arithmétique qui calcule le PGCD de plusieurs entrées. Nous donnons ensuite plusieurs algorithmes de tri et de recherche (souvent enseignés dans les cours d'algorithmique de base) en détaillant un peu plus l'algorithme de tri rapide aussi nommé `Quicksort`, très utilisé en pratique.

Algorithme naïf pour le PGCD de plusieurs entrées

Le calcul de PGCD est souvent considéré comme la cinquième opération arithmétique avec l'addition, la soustraction, la multiplication et la division. Abondamment utilisée lors de la simplification des fractions rationnelles, cette opération est aussi fondamentale dans certaines applications comme le calcul de bases de Gröbner où le calcul de PGCD représente une large part du temps d'exécution (voir [51]).

Dans cet article, nous nous intéresserons aux algorithmes du PGCD dont les entrées sont des entiers ou des polynômes sur un corps fini à q éléments $\mathbb{F}_q$. L'algorithme le plus connu est certainement l'algorithme d'Euclide décrit dans le livre VII des "Éléments" d'Euclide [18]. Selon Knuth, c'est "*le grand-père de tous les algorithmes car c'est le plus vieil algorithme non-trivial ayant survécu jusqu'à aujourd'hui*". L'algorithme effectue une succession de divisions

euclidiennes jusqu'à trouver un reste nul. Le dernier reste non nul est le PGCD (voir algorithme 1).

Algorithme 1 Euclide(x_1, x_2) : calcule le PGCD de x_1 et x_2

```
▷ μ(x) = deg x si x est un polynôme
▷ μ(x) = |x| si x est un entier
if μ(x_1) < μ(x_2) then
   échanger x_1 et x_2
end if
while x_2 ≠ 0 do
   Division euclidienne : x_1 = x_2 · q + r avec μ(r) < μ(x_2)
   (x_1, x_2) ← (x_2, r)
end while
return x_1
```

D'autres algorithmes utilisent des divisions différentes (centrée, binaire, paire, impaire, α- euclidienne, plus-moins, etc.) mais la structure générale reste identique. L'analyse dynamique, décrite à la section 1.2.2, a permis une compréhension fine du comportement probabiliste de la plupart de ces algorithmes qui agissent sur deux polynômes ou deux entiers [47, 48]. En revanche, les algorithmes agissant sur plusieurs entrées sont très peu étudiés.

Nous nous limitons ici au cas de l'algorithme dit "naïf" qui calcule le PGCD de ℓ entrées $x_1, \ldots, x_\ell$ avec $\ell \geq 2$ (voir algorithme 2). Décrit dans le livre de Knuth [33], c'est un algorithme "naturel" qui effectue une suite de $\ell - 1$ appels à un algorithme de PGCD sur deux entrées. Nous choisissons ici l'algorithme d'Euclide. Précisément, posons $y_1 := x_1$, alors pour $k \in [2..\ell]$, l'algorithme calcule $y_k := \text{PGCD}(x_k, y_{k-1}) = \text{PGCD}(x_1, x_2, \ldots, x_k)$. Le PGCD est donné par $y_\ell := \text{PGCD}(x_1, x_2, \ldots, x_\ell)$ et est obtenu après $\ell - 1$ phases. Une légère amélioration consiste à arrêter l'algorithme dès que le PGCD à la fin d'une phase est trivial (égal à 1 pour les entiers ou un polynôme de degré 0 dans le cas des polynômes).
Nous proposons à la section 1.4 une analyse fine de plusieurs paramètres décrivant le comportement de cet algorithme.

Algorithmes de tri et de recherche

Nous présentons ici cinq exemples d'algorithmes. Trois sont des algorithmes de tri (tri rapide `Quicksort`, tri insertion et tri à bulles) et deux sont des algorithmes de sélection (recherche « naïve » du minimum d'un tableau ainsi qu'une variante utilisant la même approche que `Quicksort`).

Algorithme 2 PGCD$(x_1, \ldots, x_\ell)$: calcule le PGCD de $x_1, \ldots, x_\ell$

```
▷ μ(x) = deg x si x est un polynôme
▷ μ(x) = |x| si x est un entier
y_1 = x_1
for i = 2 to ℓ do
   y_i =Euclide(y_{i−1}, x_i)
   if y_i est trivial (y_i = 1 ou deg y_i = 0) then
      return 1
   end if
end for
return y_ℓ
```

Le pseudo-code de ces algorithmes est donné ici. Nous l'espérons suffisamment explicatif et nous n'en détaillons pas vraiment le fonctionnement.

Par exemple, l'algorithme de tri le plus connu est certainement l'algorithme de tri rapide ou `Quicksort`, que l'on peut écrire de manière récursive (pour plus de précisions voir par exemple [43]). Cet algorithme utilise le principe « diviser pour régner ». Toutes les clefs sont comparées à la première clef du tableau utilisée comme pivot. Au cours du partitionnement, les clefs de valeur inférieure à celle du pivot sont disposées sur la gauche du tableau et celles dont la valeur est supérieure sont placées à droite. À la fin du partitionnement , le pivot est situé à sa place définitive. Ensuite l'algorithme `QuickSort` trie récursivement les deux sous-tableaux.

Algorithme 3 `InsSort`(V, n)

```
Ensure: Sorts the array V[1..n] (Insertion sort)
  for i from 2 to n do
     for j from i to 2 do
        if V[j − 1] ≥ V[j] then
           swap(V[j], V[j − 1])
        else
           break        ▷ exit the inner loop
        end if
     end for
  end for
```

Algorithme 4 `Quicksort`$(V, \texttt{left}, \texttt{right})$

Ensure: Sorts the subarray $V[\texttt{left}..\texttt{right}]$
  ▷ Recursive function to be called for an array $V[1..n]$: `Quicksort`$(V, 1, n)$
  ▷ `Partition`(V, i, j) rearranges the subarray $V[i..j]$ according to its first element $V[i]$, called the pivot, and returns the position of the pivot after partitionning
  if $\texttt{left} < \texttt{right}$ **then**
    $k \leftarrow$ `Partition`$(V, \texttt{left}, \texttt{right})$
    `Quicksort`$(V, \texttt{left}, k-1)$
    `Quicksort`$(V, k+1, \texttt{right})$
  end if

Algorithme 5 `BubSort`(V, n)

Ensure: Sorts the array $V[1..n]$ (Bubble sort)
  for i **from** 1 **to** $n-1$ **do**
    for j **from** n **downto** $i+1$ **do**
      if $V[j-1] > V[j]$ **then**
        `swap`$(V[j-1], V[j])$
      end if
    end for
  end for

Algorithme 6 `SelMin`(V, n)

Ensure: Returns the minimum of the array $V[1..n]$ (naive)
  `Min` $\leftarrow V[1]$
  for i **from** 2 **to** n **do**
    if $V[i] <$ `Min` **then**
      `Min` $\leftarrow V[i]$
    end if
  end for
  return `Min`

Algorithme 7 `QuickMin`(V, n)

Ensure: Returns the minimum of the array $V[1..n]$ (variation of Quickselect)
  if $n = 1$ **then**
    return $V[1]$
  end if
  `right` $\leftarrow n$
  $k \leftarrow$ `Partition`$(V, 1, \texttt{right})$ ▷ `Partition` was defined in `Quicksort`
  if $k = 1$ **then**
    return $V[1]$
  else
    return `QuickMin`$(V, 1, k-1)$
  end if

1.2 Méthodes

1.2.1 Combinatoire analytique

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 $n \geq 0$.

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

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

On note $[z^n]C(z) = C_n$. L'étude des séries génératrices de langages fait plus naturellement intervenir des séries génératrices ordinaires. Une autre forme de série génératrice, dite exponentielle, est plus adaptée aux structures dites étiquetées mais nous ne les utilisons dans ce chapitre.

La série $C(z)$ est univariée et ne permet que d'énumérer les classes d'objets. L'analyse d'algorithmes 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 $\mathcal{C}_{n,k} = \{\gamma \in \mathcal{C}_n; c(\gamma) = k\}$ et $C_{n,k} = \text{Card}\, \mathcal{C}_{n,k}$, on obtient alors des séries bivariées à deux variables

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

Remarquons qu'on a toujours $C(z) = C(z,1)$.

La valeur moyenne du paramètre c sur les données de taille n (ici en supposant une distribution uniforme sur $\mathcal{C}_n$) s'obtient par dérivation de la série génératrice bivariée par rapport à la variable u, aussi appelée série génératrice cumulée

$$\widehat{C}(z) := \frac{\partial C}{\partial u}(z,u)|_{u=1}, \quad \text{et} \quad \mathbb{E}[c_n] = \frac{1}{C_n} \sum_{k \geq 0} k C_{n,k} = \frac{[z^n]\widehat{C}(z)}{[z^n]C(z)}. \tag{1.1}$$

En dérivant encore, on a accès à la variance ou encore aux moments de la variable aléatoire. La distribution de probabilité du coût c peut être étudiée avec la série génératrice $C(z,u)$, à l'aide de la relation

$$\mathbb{P}_n[c_n = i] = \frac{[z^n\, u^i]C(z,u)}{[z^n]C(z)}.$$

Pour terminer, les probabilités $\mathbb{P}[c_n \geq m]$ s'expriment à l'aide de la "série génératrice bivariée cumulée" définie par

$$\widehat{C}^{[m]}(z) := \sum_{i \geq m} [u^i] C(z,u), \quad \text{et} \quad \mathbb{P}[c_n \geq m] = \frac{[z^n]\widehat{C}^{[m]}(z)}{[z^n]C(z)}. \tag{1.2}$$

Les relations précédentes découlent toutes de la définition formelle des séries génératrices associées au coût c et à la taille sous-jacente. La méthode symbolique vise à donner des expressions aux séries génératrices en exploitant la structure du problème.

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, nous rappelons informellement le principe de la méthode symbolique uniquement pour le cas des classes combinatoires « non étiquetées ».

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).

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

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

L'opérateur SEQ est omniprésent lors de l'analyse des algorithmes du PGCD puisque généralement, ce sont les quasi-inverses (ici $1/(1 - A(z))$) qui ont un rôle majeur d'un point de vue analytique.

Analyse complexe

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 0 redonne exactement les coefficients de la série. On se référera utilement à [25] 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 1.2.1 (Cauchy)**.** *Soit $C(z)$ un fonction analytique dans une région $\mathcal{V}$ simplement connexe de $\mathbb{C}$ contenant l'origine et Γ une courbe simple fermée, orientée positivement, à l'intérieur de $\mathcal{V}$, qui encercle 0. On a*

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

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

Plusieurs théorèmes de transfert existent et nous renvoyons à [25] pour une présentation très complète. Au chapitre 1.4, nous utilisons par exemple le résultat de transfert suivant.

Lemme 1.2.2. *Considérions une fonction $f(z) = g(z)/(1-qz)^j$ avec $j \geq 2$ et une fonction $g(z)$ qui est analytique dans le disque $|z| > 1/q$ vérifiant $g(1/q) \neq 0$. Alors,*

$$[z^n]f(z) = g\left(\frac{1}{q}\right)\binom{n+j-1}{j-1}q^n - g'\left(\frac{1}{q}\right)\binom{n+j-2}{j-2}q^{n-1} + O\left(n^{j-3}q^n\right).$$

Le résultat précédent reste vrai si g admet des pôles simples isolés sur le cercle "pointé" $\{|z| = 1/q, z \neq 1/q\}$, dès que $j \geq 3$.

Nous rencontrerons d'autres transformations intégrales (comme la formule de Rice [37, 38]) qui permettent d'exprimer des propriétés asymptotiques des coefficients des fonctions considérées.

1.2.2 Analyse dynamique

Principalement développée depuis le milieu des années 90 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. Dans cette section, nous présentons de manière succincte l'analyse dynamique et nous renvoyons à [48] pour une présentation plus complète. L'analyse dynamique est une méthode générale qui s'applique à toute une classe d'algorithmes du PGCD sur les entiers. La section 1.4, consacrée à l'analyse de l'algorithme naïf, s'appuie sur cette méthode.

Analyse dynamique et combinatoire analytique

Un point clé de la combinatoire analytique est la méthode symbolique qui transforme des opérations sur les structures combinatoires en des opérations sur les séries génératrices. Cette phase cruciale s'appuie sur des propriétés d'additivité de la taille et du paramètre étudié. Malheureusement, la taille

et/ou les paramètres lors de l'analyse des algorithmes du PGCD sur les entiers n'ont plus nécessairement ces propriétés ce qui rend la méthode symbolique difficile à appliquer. L'analyse dynamique utilise un chemin détourné qui est équivalent à la méthode symbolique mais qui s'appuie sur des outils des systèmes dynamiques. Nous en détaillons maintenant les principales étapes.

Séries génératrices

Nous conservons ici les notations de la section 1.2.1. La classe combinatoire $\mathcal{C}$ est munie d'une application taille notée $|\cdot| : \mathcal{C} \to \mathbb{N}$ telle que l'ensemble $\mathcal{C}_n = \{\gamma \in \mathcal{C}; |\gamma| = n\}$ des objets de taille n est fini pour tout $n \geq 0$ de cardinal $C_n = \text{Card}(\mathcal{C}_n)$. Les analyses dynamiques dans le cadre des sources ou des algorithmes du PGCD utilisent essentiellement des séries génératrices de type Dirichlet. La série génératrice de Dirichlet $C(s)$ associée à l'ensemble $\mathcal{C}$ est la série formelle

$$C(s) = \sum_{\gamma \in \mathcal{C}} \frac{1}{|\gamma|^s} = \sum_{n \geq 1} \frac{C_n}{n^s}.$$

Les séries de Dirichlet sont bien adaptées lorsque la taille satisfait une propriété multiplicative. C'est le cas avec les entiers puisque pour deux entiers γ et δ, $|\gamma\delta|^s = |\gamma|^s |\delta|^s$. Dans le cas d'une source sans mémoire, si p_w désigne la probabilité d'émettre le mot w, alors la probabilité d'émettre le mot $w_1 \cdot w_2$ (concaténation des mots w_1 et w_2) vérifie la propriété multiplicative $p^s_{w_1 \cdot w_2} = p^s_{w_1} p^s_{w_2}$. Nous retrouverons des exemples de séries de type Dirichlet à la fois pour l'analyse de l'algorithme naïf du PGCD et pour l'analyse des algorithmes de tri dans les sections1.4.5 et 1.3.5.

Comme pour les séries ordinaires, la série univariée $C(s)$ ne permet que d'énumérer les objets de taille donnée. Pour l'analyse d'un paramètre c, il est préférable d'utiliser la série génératrice bivariée

$$C(s,u) = \sum_{\gamma \in \mathcal{C}} \frac{u^{c(\gamma)}}{|\gamma|^s} = \sum_{n,k \geq 0} C_{n,k} \frac{u^k}{|\gamma|^s},$$

où $\mathcal{C}_{n,k} = \{\gamma \in \mathcal{C}_n; c(\gamma) = k\}$ et $C_{n,k} = \text{Card}\, \mathcal{C}_{n,k}$. Remarquons qu'on a toujours $C(s) = C(s,1)$.

Pour les séries de Dirichlet, il est souvent uniquement possible d'étudier la somme des coefficients pour $n \leq N$. C'est pourquoi nous introduisons l'ensemble $\Omega_N = \cup_{n \leq N} \mathcal{C}_n$ des entrées dont la taille est au plus N et dont le cardinal est donné par

$$\text{Card}\, \Omega_N = \sum_{n \leq N} C_n.$$

Les ensembles Ω_N sont munis de la distribution uniforme. Dans ce cas, l'espérance d'un coût c sur Ω_N, dont la restriction à Ω_N est notée c_N, s'obtient par dérivation de la série génératrice bivariée par rapport à la variable u, aussi appelée série génératrice cumulée,

$$\widehat{C}(s) := \frac{\partial C}{\partial u}(s,u)|_{u=1} \quad \text{et} \quad \mathbb{E}[c_N] = \frac{\sum_{n\leq N}[n^{-s}]\widehat{C}(s)}{\sum_{n\leq N}[n^{-s}]C(s)}, \tag{1.3}$$

où $[n^{-s}]C(s)$ désigne le coefficient C_n de la série $C(s)$. De même, $[n^{-s}u^k]C(s,u)$ désigne le coefficient $C_{n,k}$ de $C(s,u)$. En dérivant encore, on a accès à la variance ou encore aux moments de la variable aléatoire.

De manière similaire aux séries ordinaires, la distribution de probabilité du coût c peut être étudiée avec la série génératrice $C(s,u)$, via la relation

$$\mathbb{P}[c_N = i] = \frac{\sum_{n\leq N}[n^s\, u^i]C(s,u)}{\sum_{n\leq N}[n^s]C(s)}.$$

Pour terminer, les probabilités $\mathbb{P}[c_N \geq m]$ s'expriment à l'aide de la "série génératrice bivariée cumulée" définie par

$$\widehat{C}^{[m]}(s) := \sum_{i\geq m}[u^i]C(s,u), \quad \text{et} \quad \mathbb{P}[c_N \geq m] = \frac{\sum_{n\leq N}[n^s]\widehat{C}^{[m]}(s)}{\sum_{n\leq N}[n^s]C(s)}. \tag{1.4}$$

Analyse complexe et extraction des coefficients

Comme dans le cas des séries ordinaires, le comportement des séries de Dirichlet autour des singularités dominantes (ici, la singularité ayant la plus grande partie réelle) est lié à l'asymptotique des coefficients. En particulier, la position et la nature de la singularité jouent un rôle déterminant.

Cependant, ce transfert est plus difficile à obtenir pour les séries de Dirichlet. L'outil de base reste la formule de Cauchy mais ici, les cercles centrés en 0 (cas des polynômes) sont remplacés par des droites verticales non bornées. Pour cette raison, nous utilisons ici le théorème taubérien de Delange [13], qui ne concerne que les séries de Dirichlet à termes positifs et qui ne donne pas de terme de reste.

Théorème 1.2.3 (Delange). *Soit $F(s) = \sum_{n\geq 1} a_n n^{-s}$ une série de Dirichlet à termes non négatifs telle que $F(s)$ converge pour $\Re(s) > \sigma > 0$. Supposons aussi :*

(i) $F(s)$ est analytique pour $\Re(s) = \sigma,\ s \neq \sigma$,

(ii) il existe $\gamma \geq 0$, tel que

$$F(s) = A(s)(s-\sigma)^{-\gamma-1} + C(s),$$

où A, C sont analytiques en σ, avec $A(\sigma) \neq 0$.
Alors, pour $N \to \infty$,

$$\sum_{n \leq N} a_n = \frac{A(\sigma)}{\sigma\Gamma(\gamma+1)}\, N^\sigma \log^\gamma N \quad [1 + \varepsilon(N)\,], \quad \varepsilon(N) \to 0.$$

Il est tout à fait possible d'obtenir "un extracteur" avec des termes de reste en utilisant la formule de Perron [17] combinée avec la formule de Cauchy. La formule de Perron exige des conditions plus fortes et plus difficiles à démontrer à gauche de la droite verticale $\Re s = \sigma$. Dans ce cours, nous nous limitons volontairement au théorème de Delange qui nous permettra de démontrer une version affaiblie du théorème 1.4.10 à la section 1.4.5, sans terme de reste.

Systèmes dynamiques et branches inverses

Comme pour les séries ordinaires, les opérations sur les structures combinatoires peuvent se traduire en des opération sur les séries de Dirichlet. En revanche, il est nécessaire que la taille vérifie une propriété multiplicative de la forme $|(x,y)| = |x||y|$. Malheureusement, cette propriété n'existe pas lors de l'analyse des algorithmes du PGCD et la méthode symbolique doit être adaptée à ce contexte. C'est à ce stade qu'intervient l'analyse dynamique.

La première phase d'une analyse dynamique consiste à voir l'algorithme comme un système dynamique. Dans ce cours, nous nous limitons aux systèmes dynamique de l'intervalle. Un système dynamique de l'intervalle est un couple (I, S) avec I un intervalle (généralement $[0,1]$) et S une application de I dans lui-même. L'intervalle I est muni d'une partition topologique $(I_m)_{m\in\Sigma}$ avec Σ un ensemble fini ou dénombrable. La restriction de S à chaque intervalle de la partition est supposée $\mathcal{C}^2$ et strictement monotone par morceaux. Pour tout élément $m \in \Sigma$, l'image par la fonction de décalage S de l'intervalle I_m est un intervalle J_m. La restriction $S : I_m \to J_m$ définit une bijection dont la bijection inverse, notée $h_m : J_m \to I_m$, est appelée branche inverse. Nous noterons $\mathcal{H}$ l'ensemble des branches inverses.

L'algorithme d'Euclide est intimement lié au système dynamique des fractions continues. Précisément, si $a = b \cdot q + r$ est une division euclidienne, alors les rationnels $y = \frac{r}{b}$ et $x = \frac{b}{a}$ appartiennent à l'intervalle $I = [0,1]$ et

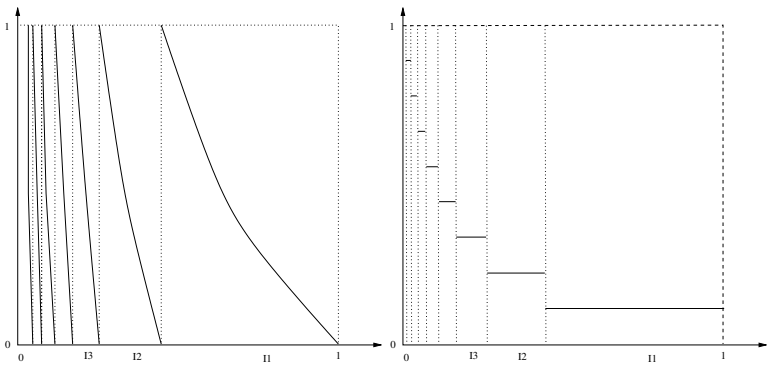


FIGURE 1.1 – *Exemple de mécanisme de source associée au développement en fraction continue. A gauche, la transformation $S_{FC}(x) = \{1/x\}$. A droite, la fonction de codage qui sera décrite dans la section consacrée aux sources (section 1.3).*

vérifient $y = S_{FC}(x)$ avec

$$S_{FC}(x) = \left\{ \frac{1}{x} \right\},$$

et $\{x\}$ la partie fractionnaire de x. Autrement dit, une étape de l'algorithme d'Euclide correspond à une itération du système dynamique des fractions continues. La fonction de décalage S_{FC} est $\mathcal{C}^2$, strictement croissante et surjective sur chaque intervalle de la partition $(I_m)_{m\in\mathbb{N}^\star}$ avec

$$I_m = \left]\frac{1}{m+1}, \frac{1}{m}\right] \quad \text{et} \quad h_m : I \to I_m, \quad h_m(x) = \frac{1}{m+x}.$$

Une représentation de S_{FC} est donnée par la figure 1.1.

Notez que les branches inverses sont toutes des homographies de déterminant 1. Comme la composée de deux homographies de déterminant 1 est également une homographie de déterminant 1, l'ensemble des branches inverses de profondeur k, notées $\mathcal{H}^k = \{g_1 \circ \cdots \circ g_k \ : \ \forall i,\ g_i \in \mathcal{H}\}$, est également formé d'homographies de déterminant 1. Nous posons $\mathcal{H}^\star$ l'ensemble de toutes les branches inverses $\mathcal{H}^\star = \cup_{k\geq 0}\mathcal{H}^k$.

Pour une entrée (u, v) avec $u > v$, l'algorithme d'Euclide (voir l'algorithme 1) calcule une suite de quotients $(m_1, \ldots, m_p)$ et construit le dévelop-

pement en fraction continue propre de v/u donné par

$$\frac{v}{u} = \cfrac{1}{m_1 + \cfrac{1}{m_2 + \cfrac{1}{\ddots \cfrac{}{m_{p-1} + \cfrac{1}{m_p}}}}} = h_{m_1} \circ h_{m_2} \circ \cdots \circ h_{m_{p-1}} \circ h_{m_p}(0),$$

avec $m_p \geq 2$. Il existe aussi un développement impropre de v/u donné par

$$\frac{v}{u} = h_{m_1} \circ h_{m_2} \circ \cdots \circ h_{m_{p-1}} \circ h_{m_p - 1} \circ h_1(0).$$

En outre, si $v/u = h \circ g(0)$ avec $h \in \mathcal{H}$, $g \in \mathcal{H}^\star$ pour un certain k, et compte tenu que g et h sont des homographies de déterminant 1, nous avons les relations fondamentales pour l'analyse des algorithmes du PGCD

$$\frac{1}{u^2} = \frac{1}{\mathrm{PGCD}(u,v)^2}(h \circ g)'(0) \quad \text{et} \quad \frac{1}{v^2} = \frac{1}{\mathrm{PGCD}(u,v)^2} g'(0). \tag{1.5}$$

Opérateurs de transfert

L'évolution des données au cours de l'algorithme d'Euclide est très liée à la dynamique du système sous-jacent. L'outil principal qui décrit l'évolution d'un système dynamique est le transformateur de densité, qui s'exprime en fonction des branches inverses comme

$$\mathbf{G}[f](t) = \sum_{m \in \Sigma} |h'_m(t)| f \circ h_m(t) \mathbf{1}_{J_m}(x). \tag{1.6}$$

Intuitivement, si f est la densité de x sur l'intervalle I, alors $\mathbf{G}[f]$ est la densité de $S(x)$. Autrement dit, l'opérateur $\mathbf{G}$ permet de suivre l'évolution des données après chaque itération du système S.

L'analyse dynamique exprime les séries génératrices en fonction des opérateurs. Les séries génératrices étant de Dirichlet, il s'avère très utile d'ajouter un paramètre complexe s au transformateur de densité. Nous obtenons l'opérateur de transfert,

$$\mathbf{G}_s[f](t) = \sum_{m \in \Sigma} |h'_m(t)|^{s/2} f \circ h_m(t) \mathbf{1}_{J_m}(x). \tag{1.7}$$

qui fut introduit par Ruelle dans [40]. Pour $h \in \mathcal{H}^\star$, l'ensemble de définition de h est noté J_h. En particulier, si $h = h_m$ pour $m \in \Sigma$, alors $J_h = J_m$. Le

k-ième itéré de $\mathbf{G}_s$ s'exprime simplement en fonction des branches inverses de profondeur k,

$$\mathbf{G}_s^k[f](t) = \sum_{h\in\mathcal{H}^k} |h'(t)|^{s/2} f\circ h(t)\mathbf{1}_{J_h}(x),$$

et le quasi-inverse $(I-\mathbf{G}_s)^{-1}$ fait intervenir toutes les branches inverses

$$(I-\mathbf{G}_s)^{-1}[f](t) = \sum_{k\geq 0} \mathbf{G}_s^k[f](t) = \sum_{h\in\mathcal{H}^\star} |h'(t)|^{s/2} f\circ h(t)\mathbf{1}_{J_h}(x).$$

Par exemple, l'opérateur de transfert associé au système dynamique des fractions continues vérifie

$$\mathbf{G}_s[f](t) = \sum_{m\geq 1} \frac{1}{(m+t)^s} f\left(\frac{1}{m+t}\right). \tag{1.8}$$

Séries génératrices et opérateurs de transfert

Une étape importante de l'analyse dynamique consiste à exprimer les séries génératrices à l'aide des opérateurs de transfert. Il existe une méthode symbolique qui transforme des opérations sur les branches inverses en opération sur les opérateurs. Ces opérations formelles sont par exemple utilisées pour l'étude de problèmes de pattern-matching [5, 6] mais dans notre cas, un calcul direct est possible.

L'idée fondamentale pour obtenir l'expression alternative des séries en fonction des opérateurs est la suivante : à tout couple d'entiers (u,v) avec $u>v>0$, on associe la branche inverse du développement en fraction continue propre h (dont le dernier quotient est nécessairement ≥ 2) et le PGCD noté y. Nous obtenons une bijection, qui combinée avec les relations 1.5, peut par exemple être utilisée pour le calcul de la série suivante

$$\sum_{u\geq 1}\sum_{0<v<u} \frac{1}{u^s} = \sum_{y\in\mathbb{N}^\star}\sum_{h\in\mathcal{H}^\star\circ\mathcal{F}} |h'(0)|^{s/2}\frac{1}{y^s} = \zeta(s)\mathbf{F}_s\circ(\mathbf{I}-\mathbf{G}_s)^{-1}[\mathbf{1}](0) \tag{1.9}$$

ou $\mathcal{F}=\{h_m : m\geq 2\}\subset\mathcal{H}$ et

$$\mathbf{F}_s[f](t) = \sum_{h\in\mathcal{F}} |h'(t)|^{s/2} f\circ h(t), \qquad \zeta(s) = \sum_{n\geq 1}\frac{1}{n^s}.$$

À la section 1.4, nous nous intéressons à une série très proche. Nous obtenons un résultat similaire (voir la proposition 1.4.8) en employant exactement les mêmes arguments et nous utiliserons en plus le développement impropre pour éviter de voir apparaître l'opérateur $\mathbf{F}_s$.

Propriétés spectrales et singularités des séries

En analyse dynamique, les singularités des séries génératrices sont principalement apportées par le quasi-inverse $(\mathbf{I} - \mathbf{G}_s)^{-1}$. D'un point de vue analytique, le quasi-inverse n'est pas défini lorsque l'opérateur $\mathbf{G}_s$ admet 1 comme valeur propre. L'analyse des singularités des séries se ramène donc à une analyse du spectre des opérateurs.

Le spectre d'un opérateur dépend de l'espace sur lequel il agit. Par exemple, le transformateur de densité $\mathbf{G}_2$ associé au système dynamique des fractions continues admet le disque unité (dans $\mathbb{C}$) comme spectre lorsqu'il agit sur l'espace de fonctions de module intégrable $L^1([0,1])$. En revanche, lorsqu'il agit sur l'espace des fonctions continuement différentiables $C^1([0,1])$, muni de la norme $||f|| = \sup_{[0,1]} |f| + \sup_{[0,1]} |f'|$, l'opérateur est quasi-compact : il admet une unique valeur propre de plus grand module (ici 1) isolée du reste du spectre par un saut spectral (voir [2]). C'est la propriété de Perron-Frobenius. La quasi-compacité des opérateurs est fondamentale pour les analyses dynamiques. L'analyse de l'algorithme du PGCD "plus-moins" est encore un problème ouvert car il n'existe pas d'espace fonctionnel connu qui entraîne la quasi-compacité des opérateurs de transfert.

Les propriétés spectrales des opérateurs de transfert associés aux fractions continues sont maintenant bien connues. Pour $s > 1$, l'opérateur $\mathbf{G}_s$ est quasi-compact et admet une unique valeur propre dominante simple, réelle et positive, notée $\lambda(s)$. Par perturbation analytique, cette propriété reste vraie dans un voisinage complexe de la demi-droite réelle $\{\Re s > 1\}$. La fonction $s \to \lambda(s)$ est strictement décroissante et comme $\mathbf{G}_2$ est un transformateur de densité, $\lambda(2) = 1$. Sur chaque droite verticale $\{\Re s = \sigma\}$ avec $\sigma > 1$, le rayon spectral de $\mathbf{G}_s$ atteint son maximum sur l'axe réel pour $s = \sigma$. Toutes ces propriétés mises ensembles montrent que le quasi-inverse $(\mathbf{I} - \mathbf{G}_s)^{-1}$ est analytique sur le demi-plan $\{\Re s \geq 2\}$ sauf en $s = 2$ où il admet une singularité (car 1 est valeur propre de $\mathbf{G}_2$).

Plus précisément, comme $\mathbf{G}_s$ possède une unique valeur propre de plus grand module $\lambda(s)$ isolée du reste du spectre par un saut spectral, l'opérateur admet une décomposition spectrale de la forme

$$\mathbf{G}_s = \lambda(s)\mathbf{P}_s + \mathbf{R}_s,$$

avec $\mathbf{P}_s$ le projecteur sur l'espace propre associé à $\lambda(s)$, $\mathbf{R}_s$ l'opérateur associé au reste du spectre (de rayon spectral $< \lambda(s)$) et $\mathbf{P}_s \circ \mathbf{R}_s = \mathbf{R}_s \circ \mathbf{P}_s = 0$. Le quasi-inverse s'écrit alors

$$(\mathbf{I} - \mathbf{G}_s)^{-1} = \frac{\lambda(s)}{1 - \lambda(s)}\mathbf{P}_s + (\mathbf{I} - \mathbf{R}_s)^{-1}$$

La valeur propre $\lambda(s)$ étant simple, $s = 2$ est un pôle simple pour le quasi-inverse.

Le quasi-inverse vérifie toutes les hypothèses du théorème taubérien de Delange (Théorème 1.2.3) ce qui permet de l'appliquer par exemple à la série génératrice (1.9).

1.3 Sources de symboles

On considère ici une source 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 (i.e., un modèle qui ne soit pas « vide » et permette de mener des calculs à terme). Par ailleurs cette section s'inspire d'une section d'un livre à paraître [9].

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–, les sources dynamiques (basées sur le principe des systèmes dynamiques déjà présentés) et également un modèle complètement général de source probabilisée.

Soit $\Sigma = \{a_1, a_2, \ldots, a_r\}$ un alphabet fini [1]. Une notion très importante va jouer un rôle dans la suite, il s'agit de la probabilité qu'un mot infini $X \in \alpha^{\mathbb{N}}$ commence par un préfixe $w \in \Sigma^*$, appelée probabilité fondamentale.

Définition 1.3.1 (Probabilités fondamentales et cylindres)**.** *Supposons une mesure de probabilité $\mathbb{P}$ définie sur l'ensemble des mots infinis (à droite) $\Sigma^{\mathbb{N}}$. La probabilité fondamentale associée au préfixe fini $w \in \Sigma^*$ est*

$$p_w = \mathbb{P}(w \cdot \Sigma^{\mathbb{N}}).$$

L'ensemble des mots infinis $C_w = w \cdot \Sigma^{\mathbb{N}}$ admettant w comme préfixe fini est également appelé cylindre *associé à w. La famille $\{C_w\}_{w \in \Sigma^*}$ suffit à décrire tous les sous-ensembles de $\Sigma^{\mathbb{N}}$.*

1. L'alphabet peut souvent être considéré dénombrable infini, mais cela complique la présentation.

1.3.1 Sources sans mémoire

Le principe d'une source sans mémoire est simple. On se donne un alphabet Σ et une loi de distribution des symboles $\{p_a\}_{a\in\Sigma}$. Puis, chaque symbole a est émis avec la probabilité p_a.

Définition 1.3.2 (Source sans mémoire)**.** *Une source sans mémoire sur l'alphabet Σ est donnée par une distribution de probabilité sur les symboles $\{p_a\}_{a\in\Sigma}$, i.e., avec $\sum_{a\in\Sigma} p_a = 1$. Pour une source sans mémoire et un préfixe fini $w = w_1 w_2 \dots w_n$ la probabilité fondamentale p_w est donnée par*

$$p_w = \prod_{i=1}^{n} p_{w_i}.$$

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 française 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 le tome 1 des *Misérables* par Victor HUGO). On obtient alors un mot qui « ressemble » déjà plus d'un point de vue syntaxique à une phrase naturelle en français (l'alphabet contient cette fois-ci les lettres accentuées, les caractères de ponctuation « ;., !" ?'- »)

UEANPNAI NYO !AHNAS EERRTQSEPINÉIRVIIIVPEIVOGELDVTA EAOIELEVMAÈI,'A
TNEIE AEAO. ULNPIOAMET.

Remarques. *Cas particulier important : alphabet binaire*. Ce sont ces sources qui sont le plus souvent rencontrées dans les analyses car elles permettent de donner des résultats plus « lisibles » (car sans trop de paramètres et donc plus facilement interprétables).

— *Source binaire sans mémoire symétrique* (appelée aussi non biaisée). Cette source peut-être vue comme le développement en base deux de réels choisis uniformément dans $[0, 1]$, mais aussi la limite du modèle uniforme du modèle fini équiprobable sur $\{0, 1\}^s$ lorsque $s \to \infty$. L'alphabet est $\{0, 1\}$ et les probabilités d'émettre 0 ou 1 est $1/2$. Pour ce modèle on a donc $p_w = 1/2^{|w|}$ pour tout mot w fini.

— *Source binaire sans mémoire biaisée* avec probabilités $(p, q = 1 - p)$ d'émettre les symboles 0 et 1 (respectivement).

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.

Définition 1.3.3 (source markovienne d'ordre 1)**.** *Une source markovienne d'ordre 1 est donnée par un ensemble de probabilités initiales $\{\pi_a\}_{a\in\Sigma}$ et une matrice stochastique de transition $P = (P_{b|a})_{(a,b)\in\Sigma\times\Sigma}$ indexée par les lettres de l'alphabet en ligne et en colonne. La quantité π_a est la probabilité que la première lettre émise soit a. La quantité $p_{b|a}$ est la probabilité d'émettre la lettre b connaissant la lettre a qui vient d'être émise. La probabilité fondamentale p_w pour un préfixe fini $w = w_1 w_2 \dots w_n$ $(n > 0)$ est*

$$p_w = \pi_{w_1} \prod_{i=2}^{n} p_{w_i|w_{i-1}}.$$

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 tome 1 des *misérables*) :

> T SSU N CHOIÉT DÉTÈRRNTR DE, ANDES DER CERT ÊM. IT PAVANDENOR U UNNTINESOUITEL'A STSITE HURQUSUPA CREUIENEUE.

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

TREMESTAIT LATTEUR. IL SAINS IN ; DANTE MIENT TREST DRA ; VENDAITÉ CHAINERTENTEL VOTÉ POURE SARQUE LE ENCTIGÉ SANCE.

(Chaîne de Markov deuxième ordre).

LE MADAMNATION DE CHAPILEMENT DANS, LUNE BAIT D'AIR DES D'ELLER SOANE, PROBLEMENTE, DANS LA MAIS D'INSTRATEUR.

(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 Sources dynamiques.

Une importante sous-classe est formée par les *sources dynamiques*, étroitement liées aux systèmes dynamiques de l'intervalle [46].

On part d'un système dynamique (I, S) dont la partition topologique est $\{\mathcal{I}_\sigma\}_{\sigma\in\Sigma}$. On munit le système (I, S) d'une application de codage $\tau : \mathcal{I} \to \Sigma$ qui est égale à σ sur $\mathcal{I}_\sigma$. Pour tout élément u de I, le mot $M(u)$ est le mot qui code la trajectoire $(u, Su, S^2u, \dots)$ à travers l'application τ, $M(u) = (\tau(u), \tau(Su), \tau(S^2u), \dots)$.

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 $S(x) = \{2x\}$ ($\{\cdot\}$ étant la partie fractionnaire). Les sources dynamiques avec une application de décalage non linéaire permettent de prendre en compte tout ou partie du « passé ». Citons par exemple les sources liées aux fractions continues obtenues avec $S(x) = \{1/x\}$ (toujours avec $\{\cdot\}$ la partie fractionnaire) et dont le système est représenté par dans la figure 1.1.

En itérant cette transformation, le développement en fraction continue de x est obtenu. Ce système de numération est notamment utilisé en algorithmique géométrique pour déterminer le signe d'un déterminant 2×2.

1.3.4 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 évidemment ardue. Le modèle idéal tient compte de l'ensemble des caractères déjà émis pour choisir le prochain symbole, et, pour un alphabet Σ, il considère les ensembles $\{p_w\}_{w\in\Sigma^k}$ des probabilités d'apparition d'un préfixe w de longueur k

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

Définition 1.3.4 (Source probabilisée)**.** *Une source probabilisée est donnée par l'ensemble* $\{p_w\}_{w\in\Sigma^*}$ *avec*

$$(\forall k \geq 0) \sum_{w\in\Sigma^k} p_w = 1, \quad (\forall w \in \Sigma^*) \sum_{a\in\Sigma} p_{w\cdot a} = p_w.$$

Cette définition revient juste à dire qu'on définit grâce à la famille des p_w une probabilité sur les cylindres et donc $\mathbb{P}$.

Remarque. La source sans mémoire comme la source markovienne permettent de calculer effectivement les probabilités fondamentales $\{p_w\}_{w\in\Sigma^*}$ à partir d'une description relativement succincte.

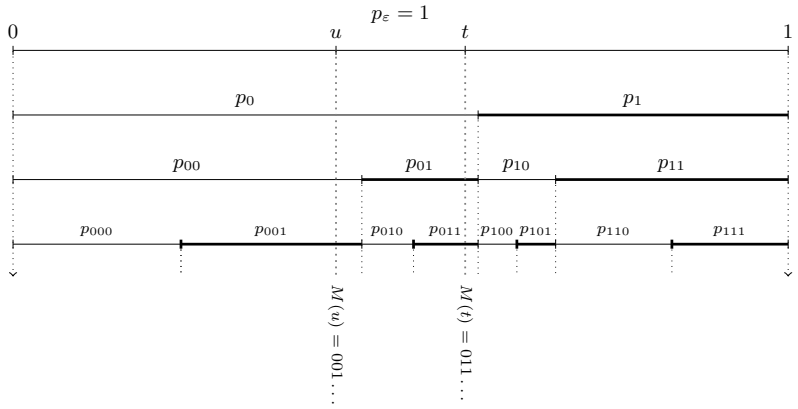


FIGURE *1.2 – Processus de raffinement pour une source binaire. On illustre également la façon dont deux mots sont associés à deux réels* u *et* t *de* $[0, 1]$.

La paramétrisation est basée sur le même principe que celui utilisé pour le codage arithmétique en compression [41]. 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 appelés intervalles fondamentaux. Cela est illustré dans la figure 1.2.

Définition 1.3.5 (Intervalle fondamental)**.** *L'intervalle fondamental associé au préfixe fini $w \in \Sigma^*$ est $\mathcal{I}_w = [a_w, b_w]$ avec*

$$a_w = \sum_{\substack{v \prec w \\ |v|=|w|}} p_v, \qquad b_w = \sum_{\substack{v \preceq w \\ |v|=|w|}} p_v = a_w + p_w, \tag{1.10}$$

où la notation '$\prec$' désigne l'ordre lexicographique sur les mots. Notons que l'intervalle $\mathcal{I}_w$ admet bien comme mesure p_w.

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 définit une application $M : [0, 1] \to \Sigma^{\mathbb{N}}$ qui associe (presque partout) à un réel u de l'intervalle unité $\mathcal{I} = [0, 1]$, un mot infini $M(u) \in \Sigma^{\mathbb{N}}$.

Définition 1.3.6 (application M)**.** *L'application M est définie presque partout par*

$$\begin{array}{rccl} M: & [0,1] & \to & \Sigma^{\mathbb{N}} \\ & u & \mapsto & M(u) = (m_1(u), m_2(u), m_3(u), \ldots), \end{array}$$

où l'application m_i est 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 $u \in \mathcal{I}_p$.

Remarque. L'application est définie presque partout car elle peut être définie de plusieurs façons aux points extrémités des intervalles [2]. L'ensemble de ces points est de mesure nulle et n'aura donc pas d'influence sur l'analyse.

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 [3] contenant u. Le processus est répété pour les symboles suivants en considérant les raffinements successifs des intervalles. Par construction pour une variable aléatoire uniforme sur $[0, 1]$ on a

$$\mathbb{P}(M(U) \text{ commence par } w) = p_w.$$

2. On peut penser par analogie aux développements impropres par exemple en base 10 : $0,999\cdots = 1,0000$ our pour les fractions continues.

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

Proposition 1.3.7 (Codage de $\Sigma^{\mathbb{N}}$ et mesure image)**.** *Soit $\mathcal{S}$ une source probabilisée donnée par la famille $(p_w)_{w\in\Sigma^*}$ et $\mathbb{P}$ la mesure de probabilité associée. On considère les intervalles fondamentaux $(\mathcal{I}_w)_{w\in\Sigma^*}$ et l'application M définie presque partout (voir définitions 1.3.5 et 1.3.6). Alors la mesure image de la mesure de Lebesgue λ sur $[0,1]$ par M est exactement $\mathbb{P}$.*

Démonstration. La mesure image de λ par M, notée pour l'instant $M(\lambda)$ est définie pour n'importe quel sous ensemble B de $\Sigma^{\mathbb{N}}$ par

$$M(\lambda)(B) = \lambda(M^{-1}(B)).$$

Soit $w \in \Sigma^*$ et $C_w = w \cdot \Sigma^{\mathbb{N}}$ le cylindre associé. On a

$$\begin{aligned} M(\lambda)(C_w) &= \lambda(M^{-1}(C_w)) \\ &= \lambda(\{u \mid M(u) \in C_w\}) \\ &= \lambda(I_w) \\ &= p_w. \end{aligned}$$

Comme la famille des C_w suffisent à décrire tous les sous-ensembles de $\Sigma^{\mathbb{N}}$, on peut donc conclure que $M(\lambda) = \mathbb{P}$. □

Cette proposition permet de paramétrer par l'intervalle $[0,1]$ les mots infinis : *tirer uniformément un mot infini de $\Sigma^{\mathbb{N}}$ pour une source probabilisée $\mathcal{S}$ (définie par ses probabilités fondamentales (p_w)) revient à tirer uniformément un réel u de $[0,1]$ et considérer le mot infini $M(u)$.* De plus cette paramétrisation préserve l'ordre lexicographique sur les mots ('$\prec$') par rapport à celui sur les réels de l'intervalle $[0,1]$ (i.e., $u < t \Leftrightarrow M(u) \prec M(t)$).

1.3.5 Série de Dirichlet de la source

Nos analyses font intervenir diverses séries de type « Dirichlet » de probabilités fondamentales mais les plus importantes sont

$$\Lambda_k(s) = \sum_{w\in\Sigma^k} p_w^s, \quad \Lambda(s) = \sum_{w\in\Sigma^*} p_w^s = \sum_{k\geq 0} \Lambda_k(s). \tag{1.11}$$

La série $\Lambda(s)$ est toujours non définie en $s = 1$ (puisque $\Lambda_k(1) = \sum_{w\in\Sigma^k} p_w = 1$ pour tout $k \geq 0$).

Deux grandeurs caractéristiques de la source jouent un rôle très important dans la suite. L'entropie $h(\mathcal{S})$ de la source $\mathcal{S}$ est définie comme la limite

(lorsque qu'elle existe) d'une quantité où interviennent les probabilités fondamentales

$$h(\mathcal{S}) = \lim_{k\to\infty} \frac{-1}{k} \sum_{w\in\Sigma^k} p_w \log p_w = \lim_{k\to\infty} \frac{-1}{k} \frac{d}{ds} \Lambda_k(s)_{|s=1} . \qquad (1.12)$$

La probabilité de coïncidence $c(\mathcal{S})$ est la longueur moyenne du préfixe commun de deux mots produits aléatoirement par la source

$$c(\mathcal{S}) = \sum_{w\in\Sigma^*} p_w^2 = \Lambda(2).$$

Lorsque $\Lambda(s)$ est analytique dans un domaine contenant un voisinage de $s = 1$, les propriétés de régularité de la source s'expriment grâce à celles de Λ dans ce domaine. Les théorèmes que nous avons établis considèrent des sources dites « *domestiquées* [4]» et ce à plusieurs degrés. On utilisera cette notion dans la section 1.5).

1.4 Analyse des algorithmes euclidiens

Cette section présente des travaux qui ont été publiés par Berthé, Creusefond, Lhote et Vallée [3] à la conférence ISSAC'13. L'algorithme d'Euclide est intimement lié au système dynamique des fractions continues $(I = [0, 1], S_{FC})$ décrit à la section 1.2.2. Précisément si $a = b\cdot q+r$ est une division euclidienne, alors les rationnels $y = \frac{r}{b}$ et $x = \frac{b}{a}$ vérifient $y = S_{FC}(x)$. Autrement dit, une itération de l'algorithme d'Euclide correspond à une itération du système dynamique des fractions continues. L'algorithme d'Euclide peut aussi être vu comme une source qui émet à chaque étape un symbole correspondant au quotient de la division euclidienne.

Ce point de vue s'étend à une large famille d'algorithmes du PGCD et établit un lien très fort avec les sources. L'analyse dynamique s'appuie implicitement sur ce point de vue et lors de l'analyse de l'algorithme naïf agissant sur les entiers (voir section 1.4.5), nous retrouverons des techniques similaires à celles utilisées dans le cadre des sources (voir section 1.5).

1.4.1 Contexte et précédents travaux

Les premières analyses probabilistes de l'algorithme d'Euclide sur les entiers sont dues à Heilbronn et Dixon qui ont montré, autour de 1970, que

4. En anglais on parle de « tameness ».

le nombre moyen de divisions euclidiennes est linéaire par rapport à la taille de l'entrée [29, 14]. En 1994, Hensley [30] a effectué la première analyse en distribution et a prouvé que le nombre d'itérations suit une loi limite gaussienne. Toutefois, la preuve ne s'étend pas facilement à d'autres paramètres de l'algorithme ou à d'autres algorithmes. Depuis le milieu des années 90, l'analyse dynamique a permis d'analyser toute une classe d'algorithmes euclidiens, avec une grande classe de paramètres [48]. Il est possible d'obtenir des résultats précis sur le comportement moyen des quotients, de la taille des continuants, du nombre d'itérations, etc. Akhavi et Vallée ont également analysé la complexité en bits moyenne [1]. En 2002, Baladi et Vallée [2] ont étendu la méthode afin d'obtenir les lois limites pour une large classe de coûts dits additifs et à croissance modérée, incluant le nombre d'itérations. Ce travail a ensuite été étendu pour l'analyse en distribution de la complexité en bits [34] ou l'analyse en moyenne d'un algorithme quasi-linéaire, de type diviser pour régner [7]. Les analyses dans le cas des polynômes sont plus simples et les mêmes résultats sont obtenus [4, 27, 31, 35].

Si le cas de deux entrées (polynômes ou entiers) est maintenant bien compris, la situation est très différente lorsque l'on considère plusieurs entrées. Avec deux entrées, seule la division est à choisir (euclidenne, centrée, impaire, etc.). Lorsqu'il y a plusieurs entrées, d'autres choix sont à faire : à chaque étape, quelles sont les deux entrées à utiliser pour la division : les deux premières, les deux plus grandes, deux au hasard ? Une fois la division effectuée, où placer le résultat : au début de la liste, à la fin, au hasard ? Faut-il ordonner ? etc. Ces choix compliquent les analyses et pour autant que nous le sachions, il n'existe pas d'analyse en moyenne de ces algorithmes du PGCD. Notez que l'analyse de l'algorithme naïf a été proposée comme un exercice dans le livre de Knuth (deuxième édition) [33]. Toutefois, pour une raison inconnue, l'exercice a disparu dans la troisième édition.

1.4.2 Analyse dans le cas particulier des polynômes

L'analyse dynamique s'applique à la fois dans le cadre des polynômes et des entiers. Il est tout à fait possible de présenter les analyses dans les deux contextes en une seule fois (voir par exemple [34]). Nous préférons ici dissocier les deux situations car les analyses avec les polynômes sont possibles avec les seuls outils de la combinatoire analytique. Nous commençons donc par le cadre le plus simple et la section 1.4.5 sera consacrée au cas des entiers. Les similitudes entre les deux situations seront alors mises en évidence.

Algorithmes et notations

Cette partie vise à comprendre précisément le comportement probabiliste de l'algorithme naïf agissant sur $\ell \geq 2$ polynômes à coefficients dans le corps à q éléments $\mathbb{F}_q$. Étant donné que l'algorithme est une succession de phases, il est important de décrire chaque phase d'indice k ($k \in [1..\ell - 1]$), avec les paramètres suivants :

(i) le nombre L_k de divisions effectuées au cours de la k-ème phase,

(ii) D_k le degré du PGCD y_k au début de la k-ème phase.

Nous sommes également intéressés par l'analyse des paramètres globaux :

(iii) l'algorithme peut être interrompu dès que $D_k = 0$ et Π est le nombre de phases utiles, à savoir :

$$\Pi = 0 \text{ si } D_1 = 0 \text{ et } \Pi := \max\{k | D_k > 0\} \text{ si } D_1 > 0.$$

(iv) Le nombre total de divisions de l'algorithme interrompu est défini comme

$$L = 0 \text{ si } \Pi = 0 \text{ et } L = L_1 + L_2 + \ldots + L_\Pi \text{ si } \Pi > 0.$$

Les entrées possibles sont tous les ℓ-uplets $\underline{x}$ formés de ℓ polynômes, et sans perte de généralité, nous supposerons tous les polynômes unitaires. L'ensemble des entrées est noté $\mathcal{C} = \mathcal{U}^\ell$, où $\mathcal{U}$ est l'ensemble polynômes unitaires. Ici, la taille de l'entrée $\underline{x}$ est le degré total du ℓ-uplet (et non pas le degré maximal comme c'est souvent le cas), et nous posons

$$\mathrm{d}(\underline{x}) = \mathrm{d}(x_1, x_2, \ldots, x_\ell) := \mathrm{d}(x_1) + \mathrm{d}(x_2) + \ldots + \mathrm{d}(x_\ell)$$

avec $\mathrm{d}(x)$ le degré de x. Les sous-ensembles $\mathcal{C}_n$ formés des entrées de taille n sont des ensembles finis munis de la distribution uniforme .

Dans ce contexte, les paramètres d'intérêt deviennent des variables aléatoires et nous nous intéressons à leurs caractéristiques probabilistes . A l'aide de la combinatoire analytique, nous décrivons à la fois le comportement moyen des principaux paramètres cités précédemment (les espérances) ainsi que les lois limites.

Premiers résultats d'analyse en moyenne

Le théorème 1.4.1 ci-dessous montre un comportement très différent entre la première phase et les suivantes. En moyenne, la première phase effectue un nombre d'itérations linéaire en la taille de l'entrée et le terme dominant met en jeu l'entropie $2q/(q-1)$ du système dynamique d'Euclide sur les polynômes. Même si le degré du premier PGCD est linéaire par rapport à la taille de l'entrée, le degré du PGCD après la première phase est en moyenne d'ordre constant. Ainsi, le nombre de divisions L_k effectuées par les autres phases

ainsi que les degrés D_k des PGCD suivants sont aussi en moyenne d'ordre constant.

Théorème 1.4.1 (Espérances). *Lorsque les ensembles $\mathcal{C}_n$ sont munis de la distribution uniforme, les principaux paramètres satisfont :*

(a) *L' espérance du nombre d'itérations L_1 au cours de la première phase est linéaire par rapport à la taille n et satisfait*

$$\mathbb{E}_n[L_1] = \frac{q-1}{2q}\frac{n}{\ell} + \frac{3q+1}{4q} + O\left(\frac{1}{n}\right).$$

(b) *Pour tout $k \in [2\,..\,\ell-1]$, l' espérance du nombre d'itérations L_k au cours de la k-ème phase est d'ordre asymptotiquement constant et vérifie*

$$\mathbb{E}_n[L_k] = \frac{q^k-1}{q^k-q}\left[1 + O\left(\frac{1}{n}\right)\right].$$

(c) *L'espérance du degré du polynôme x_1 (le premier* PGCD*) est linéaire par rapport à la taille n de l'entrée et satisfait*

$$\mathbb{E}_n[D_1] = \frac{n}{\ell}.$$

(d) *Pour tout $k\, in [2\,..\,\ell-1]$, l'espérance du degré D_k du* PGCD *y_k au début de la k-ème phase est asymptotiquement constant et vérifie*

$$\mathbb{E}_n[D_k] = \frac{1}{q^{k-1}-1}\left[1 + O\left(\frac{1}{n}\right)\right].$$

Lois limites

Dans cette section, les résultats présentés sont plus précis que ceux énoncés au théorème 1.4.1 et décrivent mieux la différence entre la première phase ($k=1$) et les phases suivantes. Pour $k=1$, les degrés attendus des deux premiers polynômes x_1 et x_2 sont linéaires et le nombre de divisions est étroitement lié à $\min(\mathrm{d}(x_1), \mathrm{d}(x_2))$. De fait, il est naturel de s'attendre à des lois bêta pour la première phase, plus précisément une loi bêta de paramètre $(1, \ell-1)$, puisque c'est la loi du minimum Y de $\ell-1$ variables aléatoires i.i.d. sur l'intervalle $[0,1]$, qui satisfait $\mathbb{P}[Y \geq x] = (1-x)^{\ell-1}$. Une telle loi a une densité égale à $(\ell-1)(1-x)^{\ell-2}$. Pour $\ell = 2$, c'est la loi uniforme. Pour les phases suivantes, des lois géométriques sont attendues puisque l'espérance des degrés des PGCD sont d'ordre constant.

Le théorème 1.4.2 décrit la distribution limite de L_k [assertion (a)] et D_k [assertion (b)]. À première vue, les résultats ne semblent pas dépendre fortement de l'indice k de la phase mais ce n'est pas le cas puisque les deux rapports

$$p_k := (q-1)/(q^k-1) \quad \text{(dans le cas } L\text{)}, \qquad r_k := q^{1-k} \quad \text{(dans le cas } D\text{)},$$

sont égaux à 1 pour $k = 1$ et sont strictement inférieurs à 1 pour $k \geq 2$.

Théorème 1.4.2 (Lois limites)**.** *Lorsque les ensembles $\mathcal{C}_n$ sont munis de la distribution uniforme, les principaux paramètres satisfont :*

(a) *le nombre d' itérations L_1 au cours de la première phase suit asymptotiquement une loi bêta de paramètre $(1, \ell-1)$ sur l'intervalle $[0, (q-1)/(2q)]$ alors que le nombre d'itérations L_k pour $k \geq 2$ suit asymptotiquement une loi géométrique de rapport $p_k := (q-1)/(q^k-1)$. On a*

$$\mathbb{P}_n\left[L_k > n/(k+1)\right] = 0, \qquad \textit{pour tout } k.$$

Pour tout k, la distribution de L_k satisfait quand $n \to \infty$,

$$\mathbb{P}_n[L_k > m] = \left(\frac{q-1}{q^k-1}\right)^m \left[1 + O\left(\frac{m}{n}\right)\right] \quad \textit{pour } m = o(n),$$

et pour $m/n \to c$ avec $c \in]0, \frac{1}{k+1}\frac{q^k-1}{q^k}[$

$$\mathbb{P}_n[L_k > m] = \left(\frac{q-1}{q^k-1}\right)^m \left(1 - \frac{(k+1)q^k}{q^k-1}c\right)^{\ell-1}\left[1 + O\left(\frac{1}{n}\right)\right].$$

(b) *Le degré D_1 du premier polynôme x_1 suit asymptotiquement une loi bêta de paramètre $(1, \ell-1)$ sur l'intervalle $[0,1]$, alors que le degré D_k du* PGCD *y_k au début de chaque phase suivante suit asymptotiquement une loi géométrique de rapport $r_k := q^{1-k}$. On a*

$$\mathbb{P}_n\left[D_k > n/k\right] = 0 \ \textit{ pour tout } k.$$

Pour tout k, la distribution de D_k satisfait lorsque $n \to \infty$,

$$\mathbb{P}_n[D_k \geq m] = q^{(1-k)m}\left[1 + O\left(\frac{m}{n}\right)\right] \quad \textit{pour } m = o(n),$$

et pour $m/n \to c$ avec $c \in]0, 1/k[$

$$\mathbb{P}_n[D_k \geq m] = q^{(1-k)m}\left(1-kc\right)^{\ell-1}\left[1 + O\left(\frac{1}{n}\right)\right].$$

Paramètres globaux

L'algorithme naïf s'arrête dès que le PGCD y_k est de degré 0. Posons $\Pi(x_1, \ldots, x_\ell)$ le nombre de phases "utiles". Comme l'évènement $[\Pi \geq k]$ coïncide avec l'évènement $[D_k \geq 1]$ pour $k \in [1\,..\,\ell-1]$, le théorème 1.4.2 conduit au résultat suivant.

Théorème 1.4.3. *Lorsque les ensembles $\mathcal{C}_n$ sont munis de la distribution uniforme, la loi du nombre Π de phases "utiles" vérifie $\mathbb{P}_n[\Pi \geq 0] = 1$, $\mathbb{P}_n[\Pi \geq \ell] = 0$,*

$$\mathbb{P}_n[\Pi \geq k] = q^{1-k}\left[1 + O\left(\frac{1}{n}\right)\right] \quad \textit{pour } k \in [1\,..\,\ell-1].$$

Le théorème 1.4.4 décrit le nombre total L de divisions effectuées par l'algorithme naïf. Avec les théorèmes 1.4.1 et 1.4.3, l'espérance est facile à calculer. De plus, les théorèmes 1.4.1 et 1.4.2 montrent que la variance de L_1 est d'ordre quadratique alors que la variance de L_k (pour $k \geq 2$) est d'ordre constant. La linéarité de l'espérance et l'inégalité de Markov impliquent que L suit la même loi asymptotique que L_1.

Théorème 1.4.4. *Lorsque les ensembles $\mathcal{C}_n$ sont munis de la distribution uniforme, le nombre total de divisions L effectuées par l'algorithme naïf admet comme espérance*

$$\mathbb{E}_n[L] = \frac{q-1}{2q}\frac{n}{\ell} + \frac{3q+1}{4q} + \sum_{k=2}^{\ell-1}\left[\frac{q}{q^k} + \frac{q-1}{q^k-1}\right] + O\left(\frac{\ell^2}{n}\right).$$

De plus, le paramètre L suit asymptotiquement une loi bêta de paramètre $(1, \ell-1)$ sur l'intervalle $[0, (q-1)/(2q)]$.

1.4.3 Séries génératrices

Manipulations formelles

Les démonstrations des différents résultats s'appuient sur la combinatoire analytique dont les outils principaux sont les séries génératrices. Nous utilisons une variable z_i pour marquer le degré du i-ème polynôme x_i, et la série génératrice $C(z_1, z_2, \ldots, z_\ell)$ de l'ensemble $\mathcal{C} = \mathcal{U}^\ell$, par rapport à la taille d est définie comme

$$C(z_1, z_2, \ldots, z_\ell) := \sum_{\underline{x} \in \mathcal{U}^\ell} z_1^{\mathrm{d}(x_1)}\, z_2^{\mathrm{d}(x_2)} \ldots z_\ell^{\mathrm{d}(x_\ell)}.$$

Elle est égale au produit $U(z_1)U(z_2)\dots U(z_\ell)$ où $U(z)$ est la série génératrice de l'ensemble $\mathcal{U}$ des polynômes unitaires par rapport à la taille d, à savoir

$$U(z) = \sum_{x\in\mathcal{U}} z^{\mathrm{d}(x)} = \sum_{n\geq 0} q^n z^n = \frac{1}{1-qz}.$$

La plupart du temps, nous nous limiterons au cas où toutes les variables z_i sont égales, et nous écrirons $C(z)$ au lieu de $C(z,\dots,z)$. Pour l'étude d'un paramètre (ou un coût) c sur $\mathcal{C} = \mathcal{U}^\ell$, l'outil principal est la série génératrice bivariée obtenue en introduisant une nouvelle variable u qui marque le coût c et qui est définie par

$$C(z,u) := \sum_{\underline{x}\in\mathcal{U}^\ell} z^{\mathrm{d}(\underline{x})} u^{c(\underline{x})}.$$

Les relations précédentes découlent toutes de la définition formelle des séries génératrices associées au coût c et à la taille sous-jacente. Les grandeurs probabilistes qui nous intéressent sont toutes liées aux coefficients des séries génératrices et il est un principe bien connu en combinatoire analytique : la position et la nature des singularités d'une série génératrice déterminent l'asymptotique des coefficients de la série. Il nous faut alors trouver une expression alternative des séries génératrices qui met en évidence leurs singularités.

Une autre expression pour les séries génératrices

Nous obtenons d'abord une expression alternative de la série génératrice $C(z)$ comme un produit de $\ell - 1$ facteurs, chacun d'eux décrivant une phase de l'algorithme.

Proposition 1.4.5. *La série génératrice de l'ensemble $\mathcal{C} = \mathcal{U}^\ell$ dont la taille est le degré total se décompose comme*

$$C(z) = U(z)^\ell = U(z^\ell)\cdot \prod_{k=1}^{\ell-1} T(z,z^k), \tag{1.13}$$

et fait intervenir la série génératrice d'une phase T définie par

$$T(z,t) = \frac{U(z)+U(t)-1}{1-G(zt)}, \tag{1.14}$$

la série génératrice $U(z)$ des polynômes unitaires et la série génératrice $G(z)$ des polynômes de degré strictement positif, i.e.,

$$U(z) = \frac{1}{1-qz}, \qquad G(z) = \frac{(q-1)qz}{1-qz}.$$

Démonstration. L'algorithme d'Euclide compare d'abord les degrés de x_1 et x_2. Il y a trois cas :

$$\mathrm{d}(x_1) = \mathrm{d}(x_2), \qquad \mathrm{d}(x_1) > \mathrm{d}(x_2), \qquad \mathrm{d}(x_1) < \mathrm{d}(x_2).$$

Dans le premier cas, il s'agit d'une soustraction supplémentaire, qui peut être considérée comme une division avec un quotient égal à 1.

Dans tous les cas, le PGCD $y := \mathrm{PGCD}(x_1, x_2)$ ainsi que la séquence des quotients $(m_1, m_2, \ldots, m_r)$ déterminent complètement la paire d'entrée (x_1, x_2). Plus précisément, on écrit $(x_1, x_2) = (y\,\widehat{x}_1, y\,\widehat{x}_2)$ avec $(\widehat{x}_1, \widehat{x}_2)$ premiers entre eux et l'exécution de l'algorithme d'Euclide sur la paire $(\widehat{x}_1, \widehat{x}_2)$ produit la même séquence $(m_1, m_2, \ldots, m_r)$ que la paire (x_1, x_2). Le premier quotient m_1 est unitaire (ceci est dû au fait que x_1 et x_2 sont unitaires) et le reste de la séquence $\underline{m} = (m_2, \ldots, m_r)$ est formé avec des polynômes généraux m_i (pas nécessairement unitaires) avec $\mathrm{d}(m_i) \geq 1$. Comme précédemment, le degré total de $\underline{m}$ est $\mathrm{d}(\underline{m}) = \mathrm{d}(m_2) + \ldots + \mathrm{d}(m_r)$.

Le premier quotient m_1 est unitaire et trois cas sont possibles :

(a) Si $\mathrm{d}(x_1) = \mathrm{d}(x_2)$, alors $m_1 = 1$.

(b) Si $\mathrm{d}(x_1) > \mathrm{d}(x_2)$ alors

$$\mathrm{d}(m_1) \geq 1, \quad \mathrm{d}(\widehat{x}_2) = \mathrm{d}(\underline{m}), \quad \mathrm{d}(\widehat{x}_1) = \mathrm{d}(m_1) + \mathrm{d}(\underline{m}).$$

(c) Si $\mathrm{d}(x_1) < \mathrm{d}(x_2)$ alors

$$\mathrm{d}(m_1) \geq 1, \quad \mathrm{d}(\widehat{x}_1) = \mathrm{d}(\underline{m}), \quad \mathrm{d}(\widehat{x}_2) = \mathrm{d}(m_1) + \mathrm{d}(\underline{m}).$$

Toutes ces remarques fournissent une expression alternative du produit $U(z_1)U(z_2)$. En effet, nous utilisons les deux relations

$$z_1^{\mathrm{d}(x_1)} z_2^{\mathrm{d}(x_2)} = (z_1 z_2)^{\mathrm{d}(y)} \cdot z_1^{\mathrm{d}(\widehat{x}_1)} z_2^{\mathrm{d}(\widehat{x}_2)},$$

$$\sum_{\widehat{x}_1, \widehat{x}_2} z_1^{\mathrm{d}(\widehat{x}_1)} z_2^{\mathrm{d}(\widehat{x}_2)} = \left[1 + \sum_{m_1} \left(z_1^{\mathrm{d}(m_1)} + z_2^{\mathrm{d}(m_1)}\right)\right] \left[\sum_{\underline{m}} (z_1 z_2)^{\mathrm{d}(\underline{m})}\right],$$

ainsi que les conditions décrites précédemment sur le premier quotient m_1, la séquence $\underline{m}$ et le PGCD y. Le premier facteur donne naissance à la série génératrice

$$1 + (U(z_1) - 1) + (U(z_2) - 1) = U(z_1) + U(z_2) - 1,$$

qui fait intervenir la série génératrice $U(z)$ des polynômes unitaires, tandis que le second facteur, qui fait intervenir une séquence de polynômes généraux ayant un degré strictement positif, s'exprime sous la forme du quasi-inverse $1/(1-G(z_1 z_2))$. Nous avons alors démontré la forme alternative suivante pour le produit

$$U(z_1)\,U(z_2) = U(z_1 z_2)\cdot T(z_1, z_2).$$

Lorsque nous remplaçons cette expression dans le produit total

$$U(z_1)\,U(z_2)\ldots U(z_\ell) = C(z_1, z_2, \ldots, z_\ell)$$

et que nous itérons cette transformation, nous obtenons une expression alternative de la série $C(z_1, z_2, \ldots, z_\ell)$ comme le produit de $\ell - 1$ facteurs, chacun d'eux s'appuyant sur la série génératrice T d'une phase en des points z_k et $t_k = z_1 \ldots z_k$

$$C(z_1, z_2, \ldots z_\ell) = U(t_\ell)\cdot \prod_{k=1}^{\ell-1} T(z_{k+1}, t_k).$$

Il peut s'avérer utile dans certaines études de garder toutes les variables z_i, mais ici, nous posons $z = z_1 = z_2 = \ldots = z_\ell$ et nous obtenons l'expression de la série génératrice $C(z)$ de l'équation (1.13). □

Séries génératrices des paramètres d'intérêt

Lors de l'étude du paramètre L_k (nombre d'itérations pendant la k-ème phase), nous utilisons une variable supplémentaire u qui marque chaque étape de la k-ième phase et nous obtenons la série génératrice

$$T(z, t, u) = u\frac{U(z) + U(t) - 1}{1 - uG(zt)} \qquad \text{avec } t = z^k,$$

qui remplace $T(z, z^k)$ dans l'expression (1.13) de $C(z)$.
Lors de l'analyse du paramètre D_k (degré du PGCD au début de la k-ième phase), nous utilisons encore une variable u qui marque cette fois-ci le PGCD y_k, et nous obtenons alors la série génératrice

$$U(t, u) = \frac{1}{1 - qut} \qquad \text{avec } t = z^k,$$

qui remplace $U(z^k)$ dans $C(z)$.

Proposition 1.4.6. *Pour tout $k \in [1\,..\,\ell-1]$, la série génératrice bivariée $L_k(z,u)$ liée au nombre de divisions euclidiennes pendant la k-ième phase satisfait*

$$L_k(z,u) = U(z)^\ell \cdot \frac{T(z,z^k,u)}{T(z,z^k)}.$$

Pour tout $k \in [1\,..\,\ell-1]$, la série génératrice bivariée $D_k(z,u)$ liée au degré du k-ième PGCD *y_k au début de la k-ième phase satisfait*

$$D_k(z,u) = U(z)^\ell \cdot \frac{U(z^k,u)}{U(z^k)}.$$

Avec la proposition 1.4.5, les séries $T(z,z^k,u)$ et $U(z^k,u)$ ont des expressions connues et de la proposition 1.4.6 découle une expression des séries génératrices bivariées $L_k(z,u)$ et $D_k(z,u)$

$$\frac{L_k(z,u)}{U(z)^\ell} = u\,\frac{1-G(z^{k+1})}{1-uG(z^{k+1})}, \qquad \frac{D_k(z,u)}{U(z)^\ell} = \frac{1-qz^k}{1-quz^k}. \tag{1.15}$$

Finalement, en prenant les dérivées par rapport à la variable u, nous obtenons les séries génératrices cumulées

$$\frac{\widehat{L}_k(z)}{U(z)^\ell} = \frac{1-qz^{k+1}}{1-q^2z^{k+1}}, \qquad \frac{\widehat{D}_k(z)}{U(z)^\ell} = \frac{qz^k}{1-qz^k}. \tag{1.16}$$

En extrayant dans (1.15) le coefficient de $[u^i]$ dans les séries bivariées et en prenant la somme sur $i \geq m$, nous trouvons

$$\frac{\widehat{L}_k^{[m]}(z)}{U(z)^\ell} = G(z^{k+1})^{m-1}, \qquad \frac{\widehat{D}_k^{[m]}(z)}{U(z)^\ell} = (qz^k)^m. \tag{1.17}$$

1.4.4 Extraction des coefficients et preuves des résultats

Les séries génératrices (1.16) et (1.17) sont des fractions rationnelles qui est un cadre très favorable pour une extraction simple des coefficients. Il est alors possible d'utiliser les relations (1.1) et (1.2) pour obtenir une expression exacte ou asymptotique de l'espérance et de la distribution de probabilité des paramètres D_k et L_k. L'asymptotique des coefficients d'une série est entièrement déterminée par la position et la nature de la singularité dominante de la série (la singularité la plus proche de 0). L'analyse des singularités permet alors un transfert entre le comportement d'une série génératrice, considérée comme une fonction analytique, près de sa singularité dominante et le comportement asymptotique de ses coefficients (voir section 1.2.1).

Calcul des espérances

Dans cette partie, les outils principaux sont les séries génératrices cumulées (1.16). Elles admettent toutes les deux un pôle dominant en $z = 1/q$ mais l'ordre du pôle est différent selon la phase. Pour la première phase ($k = 1$), le pôle $z = 1/q$ est d'ordre $\ell + 1$ alors que pour les autres phases ($k \geq 2$), ce pôle est d'ordre ℓ.

Nous considérons dans un premier temps le cas $k \geq 2$. Les expressions des séries génératrices cumulées $\widehat{D}_k(z)$ et $\widehat{L}_k(z)$ sont données par (1.16) et le lemme 1.2.2 s'applique avec $j = \ell$ et

$$g(z) = \frac{1 - qz^{k+1}}{1 - q^2 z^{k+1}}, \qquad g\left(\frac{1}{q}\right) = \frac{q^k - 1}{q^k - q} \quad \text{(cas } L\text{)},$$

$$g(z) = \frac{qz^k}{1 - qz^k} \qquad g\left(\frac{1}{q}\right) = \frac{1}{q^{k-1} - 1} \quad \text{(cas } D\text{)}.$$

Supposons maintenant $k = 1$. Dans ce cas, l'entier j vaut $\ell + 1$, et

$$g(z) = qz \quad \text{(cas } D\text{)}, \qquad g(z) = \frac{1 - qz^2}{1 + qz} \quad \text{(cas } L\text{)}.$$

Dans les deux cas, L et D, le lemme s'applique. Dans le cas D, nous obtenons :

$$[z^n]\widehat{D}_1(z) = q^n \binom{n + \ell - 1}{\ell}.$$

Nous remarquons que dans le cas L, la fonction g admet $z = -1/q$ comme pôle simple et nous obtenons

$$[z^n]\widehat{L}_1(z) = \frac{q-1}{2q} q^n \binom{n+\ell}{\ell} + \frac{q+3}{4} q^{n-1} \binom{n+\ell-1}{\ell-1} + O\left(n^{\ell-2} q^n\right).$$

Dans tous les cas, la normalisation par $\text{Card}(\mathcal{C}_n) = \binom{n+\ell-1}{\ell-1} q^n$ donne les résultats annoncés dans le théorème 1.4.1.

Un cadre général pour les lois limites

L'étude des distributions des différents paramètres se fait à l'aide des séries génératrices bivariées et cumulées, données par les formules (1.17). Ces séries sont toutes de la forme

$$U(z)^\ell \cdot A_k(z)^m,$$

où $A_k(z)$ dépend de l'indice k de la phase et du paramètre étudié. Nous avons,

$$A_k(z) = qz^k \text{ (type } D) \qquad \text{et} \tag{1.18}$$

$$A_k(z) = \frac{(q-1)qz^{k+1}}{1-qz^{k+1}} = G(z^{k+1}) \text{ (type } L). \tag{1.19}$$

Dans tous les cas, les séries génératrices s'expriment comme le produit de la fonction $U(z)^\ell$ qui admet un pôle d'ordre ℓ en $z = 1/q$, avec une "grande puissance" d'une fonction $A(z)$. Le terme "grande puissance" est utilisé car l'exposant m peut dépendre de la taille n de l'entrée. Comme le montre la proposition suivante, il existe deux cas selon que la valeur de A en la singularité dominante $z = 1/q$ est 1 ou pas. Le premier cas a lieu lorsque $k = 1$ et conduit à une loi limite bêta, alors que le second cas intervient lorsque $k \geq 2$ et conduit à des lois géométriques.

Proposition 1.4.7. *Considérons la série génératrice*

$$F^{[m]}(z) = \frac{1}{(1-z)^\ell} \cdot A(z)^m,$$

où $A(z)$ est analytique sur le disque $|z| \leq \rho$ avec $\rho > 1$ et satisfait $a := A(1) \neq 0$, $b := A'(1) > 0$ et pour $|z|$ suffisamment proche de 1, $|A(z)| \leq A(|z|)$.
Alors, le coefficient de z^n dans $F^{[m]}(z)$ vérifie :

(a) *lorsque $m/n \to 0$, alors*

$$[z^n]F^{[m]}(z) = \frac{n^{\ell-1}}{(\ell-1)!} \cdot a^m \left[1 + O\left(\frac{m}{n}\right)\right].$$

(b) *lorsque $m/n \to c$ pour une constante $c \in]0, a/b[$, alors*

$$[z^n]F^{[m]}(z) = \frac{n^{\ell-1}}{(\ell-1)!}\, a^m \left(1 - \frac{b}{a}c\right)^{\ell-1} \left[1 + O\left(\frac{1}{n}\right)\right].$$

Fin de la preuve du théorème 1.4.2

Les probabilités $\mathbb{P}_n[L_k > m]$, $\mathbb{P}_n[D_k > m]$ sont liées au coefficient z^n des séries génératrices décrites par les relations (1.17), et les fonctions $A_k(z)$ sont données selon les cas par les formules (1.18) et (1.19). Puisque dans le cas D, $A_k(z)$ est multiple de z^k, et dans le cas L, $A_k(z)$ est multiple de z^{k+1}, nous obtenons dans un premier temps les égalités suivantes

$$\mathbb{P}_n\left[D_k > n/k\right] = 0, \qquad \mathbb{P}_n\left[L_k > n/(k+1)\right] = 0.$$

Avec le changement de variable $z \to z/q$, les hypothèses de la proposition 1.4.7 sont toutes satisfaites pour $z \mapsto A_k(z/q)$, et nous obtenons,

$$a_k = q^{1-k}, \qquad \frac{b_k}{a_k} = k \qquad \text{(cas } D\text{)},$$

$$a_k = \frac{q-1}{q^k-1}, \qquad \frac{b_k}{a_k} = (k+1)\frac{q^k}{q^k-1} \qquad \text{(cas } L\text{)}.$$

Pour $k = 1$, les constantes a_k valent 1, alors qu'elles sont strictement plus petites que 1 pour $k \geq 2$. Finalement, la normalisation par

$$\text{Card}(\mathcal{C}_n) = \frac{n^{\ell-1}}{(\ell-1)!} q^n \left[1 + O\left(\frac{1}{n}\right)\right],$$

prouve les assertions (a) et (b) du théorème 1.4.2.

1.4.5 Analyse pour des entrées entières

Nous expliquons maintenant brièvement comment une étude similaire peut être effectuée dans le cas de l'algorithme naïf sur des nombres entiers. Comme d'habitude (voir par exemple [34, 48]), l'étude avec les polynômes montre le chemin, et des résultats similaires sont attendus dans le cas des entiers même s'ils sont souvent plus difficiles à obtenir et moins précis.

L'algorithme naïf sur les entiers

Dans le cas des entiers, l'algorithme naïf a exactement la même structure que dans le cas des polynômes. Il est composé de $\ell-1$ phases, chacune d'elles appelant l'algorithme d'Euclide qui effectue le calcul de PGCD entre deux entiers. Son exécution est décrite par l'algorithme 2. Les entrées possibles sont tous les ℓ-uplets $\underline{x}$ formés de ℓ entiers, et sans perte de généralité, nous nous limitons à des entiers positifs. L'ensemble des entrées est $\mathcal{C} = \mathbb{N}^\ell$, où $\mathbb{N}$ est l'ensemble des entiers positifs. Dans le cadre des entiers, la taille est habituellement la longueur binaire des entiers définie par $\mu(x) = \lfloor \ln x \rfloor + 1$. Dans ce chapitre, il sera plus facile de définir la taille d d'un entier x par $\mathrm{d}(x) = \lceil \ln x \rfloor$, et la taille $\mathrm{d}(\underline{x})$ de l'entrée $\underline{x} = (x_1, \ldots, x_\ell)$ est $\mathrm{d}(x_1 x_2 \cdots x_\ell)$. La taille d joue le même rôle que le degré sur les polynômes mais contrairement aux polynômes, la taille n'est plus additive puisque $\mathrm{d}(x_1, \ldots, x_\ell) \neq \mathrm{d}(x_1) + \ldots + \mathrm{d}(x_\ell)$. Nous définissons aussi la longueur d'un entier x comme étant l'entier x lui même. Par extension, la longueur d'une entrée $\underline{x}$ est la longueur du produit $x_1 x_2 \cdots x_\ell$ de ses composantes.

Ici encore, le sous-ensemble $\mathcal{C}_n$ formé avec les entrées de taille au plus n est un ensemble fini, et il est doté de la probabilité uniforme.

Séries génératrices de Dirichlet

Dans le cadre des entiers, les séries génératrices sont de type Dirichlet. La série génératrice de Dirichlet associée à l'ensemble $\mathbb{N}^\ell$ des ℓ–uplets $\underline{x} = (x_1, x_2, \ldots, x_\ell)$ d'entiers est donnée par

$$C(s_1, s_2, \ldots, s_\ell) = \sum_{\underline{x} \in \mathbb{N}^\ell} \frac{1}{x_1^{s_1}} \frac{1}{x_2^{s_2}} \cdots \frac{1}{x_\ell^{s_\ell}} = \zeta(s_1) \ldots \zeta(s_\ell),$$

où la fonction ζ est la série génératrice de l'ensemble $\mathbb{N}$,

$$\zeta(s) := \sum_{n \geq 1} \frac{1}{n^s}.$$

Comme dans le cas des polynômes, nous serons intéressés au cas où $s_1 = s_2 = \ldots s_\ell = s$ et nous obtenons

$$C(s) := C(s, \ldots, s) = \zeta(s)^\ell,$$

Cependant, pour trouver une forme alternative aux séries génératrices, nous commençons par le cas générique avec des s_i différents.

Comme précédemment, la première étape consiste à trouver une décomposition du produit $\zeta(s_1)\,\zeta(s_2)$ de la forme

$$\zeta(s_1)\,\zeta(s_2) = \zeta(s_1 + s_2) \cdot T(s_1, s_2),$$

où $T(s_1, s_2)$ est la série génératrice d'une phase qui décrit l'exécution de l'algorithme d'Euclide agissant sur deux entiers. L'évolution de l'algorithme d'Euclide se décrit à l'aide de l'opérateur de transfert donné par la formule 1.8 et associé au système dynamique des fractions continues (voir section 1.2.2). Le résultat suivant est l'analogue de la proposition 1.4.5

Proposition 1.4.8. *La série génératrice de $\mathcal{C} = \mathbb{N}^\ell$ avec la taille "produit des entrées" peut s'écrire comme*

$$C(s) = \zeta(s)^\ell = \zeta(\ell s) \cdot \prod_{k=1}^{\ell-1} T(s, ks), \tag{1.20}$$

et met en jeu la série génératrice d'une phase T définie par

$$T(s,t) = \frac{\zeta(s)\zeta(t)}{\zeta(s+t)} = \frac{1}{2}\left[(I - \mathbf{G}_{s+t})^{-1} \circ (\mathbf{G}_s + \mathbf{G}_t)[1](0)\right], \tag{1.21}$$

où $\mathbf{G}_s$ est l'opérateur de transfert associé au système dynamique d'Euclide, défini par l'équation (1.8).

Démonstration. L'algorithme d'Euclide compare d'abord les deux entiers x_1 et x_2. Il y a trois cas :

$$x_1 = x_2, \qquad x_1 > x_2, \qquad x_1 < x_2.$$

Dans tous les cas, le PGCD $y := \text{PGCD}(x_1, x_2)$ ainsi que les quotients $(m_1, m_2, \ldots, m_r)$ déterminent entièrement la paire initiale (x_1, x_2). Précisément, nous écrivons $(x_1, x_2) = (y\,\widehat{x}_1, y\,\widehat{x}_2)$ avec $(\widehat{x}_1, \widehat{x}_2)$ des nombres premiers entre eux et l'exécution de l'algorithme d'Euclide sur la paire $(\widehat{x}_1, \widehat{x}_2)$ produit la même séquence $(m_1, m_2, \ldots, m_r)$ que la paire (x_1, x_2), avec maintenant des restes $\widehat{x}_i$ qui satisfont $x_i = y\widehat{x}_i$.

L'exécution de l'algorithme d'Euclide sur la paire $(\widehat{x}_1, \widehat{x}_2)$ avec $\widehat{x}_1 > \widehat{x}_2$ construit un développement en fraction continue

$$\frac{\widehat{x}_2}{\widehat{x}_1} = h \circ g(0), \qquad \frac{\widehat{x}_3}{\widehat{x}_2} = g(0).$$

Ici, $h := h_{m_1}$ est lié au premier quotient et $g = h_{m_2} \circ h_{m_3} \circ \cdots \circ h_{m_r}$ est liée à la séquence $(m_2, m_3, \ldots, m_r)$. Comme les deux paires $(\widehat{x}_1, \widehat{x}_2)$ et $(\widehat{x}_2, \widehat{x}_3)$ sont premières entre elles, les dénominateurs de chaque rationnel s'exprime avec les dérivées,

$$\frac{1}{\widehat{x}_1^2} = |(h \circ g)'(0)| = |h'(g(0))| \cdot |g'(0)|, \qquad \frac{1}{\widehat{x}_2^2} = |g'(0)|.$$

Alors, dans ce cas lorsque $\widehat{x}_1 \geq \widehat{x}_2$, la somme

$$\sum_{\widehat{x}_1 \geq \widehat{x}_2} \frac{1}{\widehat{x}_1^{s_1}} \frac{1}{\widehat{x}_2^{s_2}} = 1 + \sum_{h,g} |h'(g(0))|^{s_1/2} \cdot |g'(0)|^{(s_1+s_2)/2},$$

s'exprime avec l'opérateur de transfert $\mathbf{G}_s$ comme [5]

$$\frac{1}{2}\left[(I - \mathbf{G}_{s_1+s_2})^{-1} \circ \mathbf{G}_{s_1}[1](0) + 1\right].$$

5. Le facteur (1/2) est ici pour tenir compte du fait que tout rationnel de $]0,1]$ admet deux développements en fraction continue : le développement propre et le développement impropre (voir section 1.2.2)

Le cas $\widehat{x}_2 \geq \widehat{x}_1$ peut être traité en échangeant les rôles de $\widehat{x}_1$ et $\widehat{x}_2$. Finalement, les relations

$$\sum_{x_1,x_2} \frac{1}{x_1^{s_1}} \frac{1}{x_2^{s_2}} = \sum_{v} \frac{1}{y^{s_1+s_2}} \sum_{\widehat{x}_1,\widehat{x}_2} \frac{1}{\widehat{x}_1^{s_1}} \frac{1}{\widehat{x}_2^{s_2}}$$

entraînent l'égalité

$$\zeta(s_1)\zeta(s_2) = \zeta(s_1+s_2) \cdot T(s_1,s_2),$$

où T est définie en (1.21). Lorsque l'on insère cette expression dans le produit total,

$$\zeta(s_1)\,\zeta(s_2)\ldots\zeta(s_\ell) = C(s_1,s_2,\ldots,s_\ell),$$

et que l'on itère la transformation, nous obtenons une expression alternative de la série génératrice $C(s_1,s_2,\ldots,s_\ell)$ comme un produit de $\ell-1$ facteurs, chacun d'entre eux faisant intervenir la série génératrice d'une phase T aux points s_k et $t_k = s_1+\cdots+s_k$,

$$C(s_1,s_2,\ldots,s_\ell) = \zeta(t_\ell) \cdot \prod_{k=1}^{\ell-1} T(s_{k+1},t_k).$$

Comme dans le cas des polynômes, il peut être parfois utiles pour certaines études de conserver toutes les variables s_i, mais ici nous posons encore $s_1 = s_2 = \ldots = s_\ell = s$, et nous obtenons l'expression attendue pour la série génératrice $C(s)$. □

Séries génératrices de Dirichlet des paramètres d'intérêt

Pour l'analyse du paramètre L_k (nombre d'itérations lors de la k-ième phase), nous ajoutons une variable u qui marque les itérations de la k-ème phase et nous utilisons la série génératrice

$$2T(s,t,u) = u(1-u\mathbf{G}_{s+t})^{-1} \circ (\mathbf{G}_s + \mathbf{G}_t)[1](0),$$

avec $t = ks$, qui remplace $2T(s,ks)$ dans l'équation (1.20) de $C(s)$.
Lorsque nous étudions le paramètre D_k (taille du PGCD au début de la k-ième phase), nous utilisons aussi une variable supplémentaire u qui marque cette fois-ci la taille du PGCD y_k, et nous utilisons une série génératrice de type zeta

$$\zeta(t,u) = \sum_{n\geq 1} \frac{u^{\mathrm{d}(n)}}{n^t}, \qquad \text{avec } t = ks$$

qui remplace $\zeta(ks)$ dans l'équation (1.20) de $C(s)$. Finalement, nous obtenons l'analogue de la proposition 1.4.6.

Proposition 1.4.9. *Pour tout $k \in [1\,..\,\ell-1]$, la série génératrice bivariée $L_k(s,u)$ liée au nombre de divisions pendant la k-ième phase vérifie*

$$L_k(s,u) = \zeta(s)^\ell \cdot \frac{T(s,ks,u)}{T(s,ks)}.$$

Pour tout $k \in [1\,..\,\ell-1]$, la série génératrice bivariée $D_k(z,u)$ liée à la taille du k-ième PGCD *y_k au début de la k-ième phase satisfait*

$$D_k(s,u) = \zeta(s)^\ell \cdot \frac{\zeta(ks,u)}{\zeta(ks)}.$$

Avec la proposition 1.4.8, la fonction $T(s,ks,u)$ admet une expression explicite. En utilisant la proposition 1.4.9, et en prenant la dérivée par rapport à u, nous obtenons les séries génératrices cumulées

$$\frac{\widehat{D}_k(s)}{\zeta(s)^\ell} = \frac{\widehat{\zeta}(ks)}{\zeta(ks)}, \qquad \frac{\widehat{L}_k(s)}{\zeta(s)^\ell} = \frac{\widehat{T}(s,ks)}{T(s,ks)}. \tag{1.22}$$

avec

$$\widehat{\zeta}(ks) = \frac{\partial \zeta}{\partial u}(ks,u)|_{u=1} \qquad \text{et} \qquad \widehat{T}(s,ks) := \frac{\partial T}{\partial u}(s,ks,u)|_{u=1},$$

qui met en jeu deux occurrences du quasi-inverse $(I-\mathbf{G}_{(k+1)s})^{-1}$ puisque

$$\frac{\partial T}{\partial u}(1-u\mathbf{G}_s)^{-1}|_{u=1} = (1-\mathbf{G}_s)^{-1} \circ \mathbf{G}_s \circ (1-\mathbf{G}_s)^{-1}.$$

Résultats d'analyse en moyenne

Le résultat qui suit est l'exact analogue du théorème 1.4.1. En particulier, dans l'assertion (a), l'entropie $\pi^2/(6\log 2)$ du système dynamique des fractions continues sur les entiers remplace son analogue polynomial $(2q)/(q-1)$ sur $\mathbb{F}_q[X]$. Nous notons aussi Ω_N l'ensemble des entrées $\underline{x}$ dont la longueur $x_1x_2\ldots x_\ell$ est plus petite que N. Cela revient à dire que la taille de $\underline{x}$ est plus petite que $\log N$.

Théorème 1.4.10 (Espérances)**.** *Lorsque l'ensemble Ω_N est muni de la distribution uniforme, les résultats suivant s'appliquent :*

(a) *L'espérance du nombre d'itérations L_1 pendant la première phase est linéaire par rapport à la longueur $\log N$ et satisfait*

$$\mathbb{E}_n[L_1] = \frac{6\log 2}{\pi^2}\left(\frac{1}{\ell}\log N\right)\left[1+O\left(\frac{1}{\log N}\right)\right].$$

(*b*) *Pour tout* $k \in [2\,..\,\ell-1]$, *l'espérance du nombre d'itérations* L_k *pendant la* k*-ème phase est asymptotiquement constant, de valeur* a_k, *qui s'exprime en fonction de l'opérateur de transfert* $\mathbf{G}_s$ *en* $s = k+1$,

$$\mathbb{E}_n[L_k] = a_k \left[1 + O\left(\frac{1}{\log N}\right)\right].$$

(*c*) *L'espérance de la longueur du premier entier* u_1 *est linéaire par rapport à la longueur* $\log N$ *et satisfait*

$$\mathbb{E}_n[D_1] = \frac{1}{\ell}\log N.$$

(*d*) *Pour tout* $k \in [2\,..\,\ell-1]$, *l'espérance de la longueur du* PGCD y_k *au début de la* k*-ième phase est asymptotiquement constant et satisfait*

$$\mathbb{E}_n[D_k] = \frac{\widehat{\zeta}(k)}{\zeta(k)}\left[1 + O\left(\frac{1}{\log N}\right)\right].$$

Idée de la preuve du théorème 1.4.10

Nous avons obtenu en (1.22) les expressions des séries génératrices cumulées. Il est maintenant possible d'extraire les coefficients de ces séries de Dirichlet en appliquant le théorème taubérien de Delange 1.2.3. Ensuite, le calcul des espérances s'obtient en appliquant la relation 1.3 qui lie les espérances aux coefficients.

Le théorème 1.2.3 s'applique avec $\sigma = 1$ aux séries génératrices cumulées définies en (1.22). Nous étudions ensuite les propriétés analytiques des séries $\zeta(s)^\ell, \widehat{\zeta}(s),\ \widehat{T}(s,ks), T(s,ks)$. Pour les deux premières, suffisamment d'informations sont connues sur les fonctions ζ pour une application directe du théorème de Delange 1.2.3 dans le cas D (voir [16]).

Nous nous intéressons maintenant au cas L, pour lequel nous utilisons des résultats récents sur l'opérateur de transfert $\mathbf{G}_s$, lorsqu'il agit sur l'espace de Banach $\mathcal{B} := \mathcal{C}^1([0,1])$. Tout d'abord, la fonction $\mathbf{G}_s[1]$ est la fonction zeta de Hurwitz qui admet un pôle d'ordre 1 en $s = 1$, et proche de $s = 1$, cette fonction vérifie $\mathbf{G}_s[1] \sim f/(s-1)$, où f appartient à l'ensemble $\mathcal{B}$. De fait, pour s proche de 1,

$$2^{\delta(k,1)}\, T(s,ks) \sim 1/(s-1)(I - \mathbf{G}_{(k+1)s})^{-1}[f](0),$$

pour tout k avec $\delta(i,j)$ est le symbole de Kronecker.

Le quasi-inverse est un objet bien connu dans le cadre de l'analyse dynamique et les propriétés spectrales de $\mathbf{G}_t$ ainsi que les propriétés analytiques du quasi-inverse ont été décrits à la section 1.2.2. En particulier pour

$\Re t > 2$, l'opérateur $\mathbf{G}_t$ a un rayon spectral strictement plus petit que 1 et le quasi-inverse $(I-\mathbf{G}_t)^{-1}$ est analytique sur ce domaine. Sur la droite verticale $\Re t = 2$, le quasi-inverse $(I - \mathbf{G}_t)^{-1}$ est analytique sauf en $t = 2$ où il admet un pôle simple avec un résidu qui fait intervenir l'entropie. Comme $T(s, ks)$ (resp. $\widehat{T}(s, ks)$) contient une (resp. deux) occurrence(s) du quasi-inverse, nous obtenons finalement :
– pour $k \geq 2$, les séries $T(s, ks)$ et $\widehat{T}(s, ks)$ admettent un pôle simple en $s = 1$,
– pour $k = 1$, la série $T(s, s)$ admet un pôle double en $s = 1$, et la série $\widehat{T}(s, s)$ admet un pôle triple en $s = 1$.
Finalement, la séries $\widehat{L}_k(s)$ satisfait les hypothèses du théorème 1.2.3, avec $\sigma = 1$, l'exposant γ vaut ℓ pour $k \geq 2$ et $\ell + 1$ pour $k = 1$. Cela conclut la preuve dans le cas des paramètres de type L.

Pour aller un peu plus loin

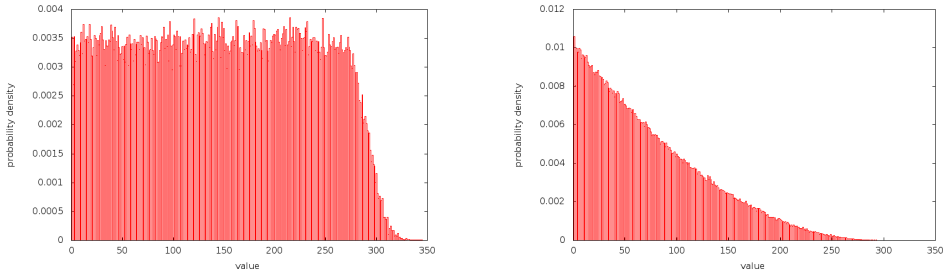
Nous pouvons aussi étudier le nombre de phases utiles et retrouver le résultat bien connu $\mathbb{P}_N[D_k = 0] \sim 1/\zeta(k)$. Il est possible d'obtenir les termes de reste dans le théorème 1.4.10 et un analogue des lois limites décrites dans le théorème 1.4.2. Il est nécessaire de prouver un résultat similaire à la proposition 1.4.7 ce qui est possible avec la formule de Perron [17], déjà utilisée dans de précédentes analyses en distribution [2, 34].

La formule de Perron fournit des termes de reste dès que (i) la série de Dirichlet possède une bande verticale sans pôle à gauche de la droite verticale $\Re s = 1$, (ii) $s = 1$ est l'unique pôle sur $\Re s \geq 1$, (iii) et la série est de croissance polynomiale pour $|\Im s| \to \infty$. Des résultats classiques [16] montrent que la fonction ζ et ses dérivées vérifient ce type de propriétés et des résultats dus à Dolgopyat-Baladi-Vallée [15, 2] montrent que ces propriétés sont aussi vérifiées pour les opérateurs de transfert $\mathbf{G}_{2s}$. Il est alors possible d'obtenir un analogue du théorème 1.4.2 lorsque n, m sont remplacés par $\log N, \log M$ et les rapports p_k, r_k sont remplacés par

$$\widehat{p}_k := \lambda\left((k+1)/2\right) \ \text{(cas } L\text{)}, \quad \widehat{r}_k := \exp(1-k) \ \text{(cas } D\text{)}.$$

Ici, $\lambda(s)$ est la valeur propre dominante de l'opérateur $\mathbf{G}_{2s}$ (lorsqu'il agit sur l'espace de fonctions $\mathcal{B}$), et joue un rôle central dans l'analyse dynamique de plusieurs algorithmes de PGCD. Elle satisfait $\lambda(1) = 1$, et pour $k = 3$, nous retrouvons la constante $\lambda(2)$ qui joue un rôle important lors de l'analyse de l'algorithme de Gauss pour la réduction des réseaux [12, 26].

Expérimentations



La figure montre les densités expérimentales de L_1 pour $\ell = 2$ (gauche) et $\ell = 4$ (droite) et met clairement en évidence la loi limite uniforme (gauche) et la loi limite béta (droite). Ces expériences ont été obtenues avec $5 \cdot 10^5$ exécutions sur des entrées entières $(x_1, \ldots, x_\ell)$ dont la longueur totale vérifie $\log_2 N = 10^4$.

1.5 Algorithmes de tri et de recherche

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 [32, 42]. 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.

Ici nous décrivons l'approche générale déjà esquissée dans [49] pour *l'analyse en moyenne d'algorithmes de tri et de recherche pour le nombre de comparai-*

sons de symboles, et présentée plus en détails dans l'article à paraître [11] (dont ce chapitre emprunte globalement la structure). Les travaux antérieurs [22, 20, 21, 19] se focalisent en général sur un algorithme spécifique (tel `Quicksort` ou `Quickselect`) le plus souvent pour une source binaire sans mémoire. On souhaite évaluer l'impact de la représentation des clefs, lorsque que celles-ci sont des mots produits par une source, sur le nombre de comparaisons au niveau des symboles. La démarche générale procède en trois étapes :

1. D'abord la stratégie de l'algorithme (quelles clefs sont comparées ? avec quelle probabilité ?) est étudiée.
2. Ensuite, nous considérons que les clefs sont produites à l'aide d'une source probabilisée, ce qui donne une expression exacte mais compliquée du nombre de comparaisons de symboles.
3. Enfin, une analyse asymptotique permet de préciser le comportement asymptotique du nombre de comparaisons de symboles.

La première étape est de nature combinatoire. La deuxième étape est plutôt algébrique et peut souvent être automatisée. La dernière étape fait appel à des techniques d'analyse complexe et suppose quelques propriétés de la source probabilisée (ou plutôt de sa série de Dirichlet).

Cette méthode s'applique avec succès aux algorithmes de tri ou de sélection présentés à la section 1.1.2 p. 4. Nous examinerons donc dans cette section cinq algorithmes classiques de base : le tri rapide `Quicksort`, le tri par insertion `InsSort`, le tri à bulles `BubSort`, et deux algorithmes de sélection du minimum d'un tableau (l'algorithme naïf `SelMin` et une variante `QuickMin` inspirée du principe de partitionnement de `QuickSort`). Ces algorithmes, extrêmement classiques et plutôt simples dans leur fonctionnement, ont été rappelés (sous forme de pseudo-code) dans la section 1.1.2.

1.5.1 La stratégie de l'algorithme

Afin de décrire la probabilité que deux clefs soient comparées, nous devons d'abord clairement poser le modèle probabiliste sous-jacent. C'est en fait le plus classique qui soit puisqu'il s'agit du modèle des permutations uniformes, auquel nous ajouterons une couche « source de symboles ».

Modèle des permutations

Considérons un ensemble totalement ordonné de clefs $\mathcal{U} = \{U_1 < U_2 < \cdots < U_n\}$ et n'importe quel algorithme $\mathcal{A}$ qui prend des décisions en comparant les clefs (mais qui ne modifie pas la *valeur* des clefs). La donnée initiale de

l'algorithme (l'entrée) est une séquence $\mathcal{V} = (V_1, V_2, \ldots, V_n)$ définie d'après $\mathcal{U}$ grâce à une permutation $\sigma \in \mathfrak{S}_n$ via les égalités $V_i = U_{\sigma(i)}$. L'exécution de l'algorithme, si elle ne dépend pas de la représentation des clefs, ne dépend pas non plus de la séquence $\mathcal{V}$ mais seulement de la permutation σ qui définit la séquence d'entrée. Ainsi, de manière classique on considère souvent dans la littérature que la permutation σ est l'entrée de l'algorithme et que l'ensemble des entrées de taille n est l'ensemble des permutations $\mathfrak{S}_n$.

Exemple.*Pour $n = 4$, une séquence (ou tableau) de clefs de l'intervalle $[0, 1]$ $\mathcal{V} = (0.44, 0.12, 0.98, 0.34\}$ correspond à un univers de clefs $\mathcal{U} = \{0.12, 0.34, 0.44, 0.98\}$. Pour la relation d'ordre, la séquence $\mathcal{V}$ est équivalente à la permutation $\sigma = (3\,1\,4\,2)$.*

Comparer une paire de clefs

La stratégie de l'algorithme $\mathcal{A}$ définit pour chaque paire (i, j), avec $1 \leq i < j \leq n$, le sous-ensemble de $\mathfrak{S}_n$ qui rassemble les permutations σ pour lesquelles les clefs U_i et U_j sont comparées au cours de l'exécution de $\mathcal{A}$, lorsque la donnée en entrée est $(U_{\sigma(1)}, U_{\sigma(2)}, \ldots, U_{\sigma(n)})$. Pour les algorithmes efficaces, deux clefs U_i et U_j sont comparées au plus une fois, mais il existe des algorithmes (disons moins intelligents, tels le célèbre algorithme du tri à bulles) où deux clefs U_i et U_j sont susceptibles d'être comparées plusieurs fois. Quoiqu'il en soit, considérant la distribution uniforme sur les permutations, $\pi_n(i, j)$ désignera toujours le nombre moyen de comparaisons entre U_i et U_j pour une séquence en entrée de taille n. Le calcul de $\pi_n(i, j)$ est la première étape de notre méthode.

Calcul de la probabilité $\pi_n(i, j)$.

L'univers des clefs (complètement ordonné) $\mathcal{U}$ étant fixé, une permutation σ définit donc une entrée de l'algorithme $\mathcal{V} = (V_1, V_2, \ldots, V_n)$, avec $V_i = U_{\sigma(i)}$. Nous noterons $\tau(U_i)$ la *position* de U_i comme la position de U_i dans la séquence $\mathcal{V}$ d'entrée. Bien sûr il y a une relation simple entre position τ dans la séquence et permutation σ puisque $\tau(U_i) = j$ si et seulement si $V_j = U_i$ (ce qui signifie également $\sigma(j) = i$).

Une propriété *remarquable* pour les cinq algorithmes de cette étude est que ces espérances $\pi_n(i, j)$ s'expriment *toujours* comme des fractions rationnelles d'une forme particulière. Cette propriété permettra d'automatiser certaines étapes du calcul lorsque l'on considérera de plus qu'une source produit l'ensemble $\mathcal{U}$ des clefs.

Nous présentons dans la figure 1.1 les expressions pour le nombre moyen $\pi_n(i, j)$ de comparaisons entre les clefs U_i et U_j, pour chaque algorithme étu-

dié. Avec ces expressions, il évidemment aisé de retrouver les expressions (classiques) pour le nombre moyen C_n de comparaisons de clefs (troisième colonne). En effet de manière générale, on a naturellement

$$C_n = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \pi_n(i,j).$$

Algorithmes	$\pi_n(i,j)$	$C(n)$
`QuickSort`	$\dfrac{2}{j-i+1}$	$2n\log n$
`InsSort`	$\dfrac{1}{2} + \dfrac{1}{(j-i+1)(j-i)}$	$\dfrac{n^2}{4}$
`BubSort`	$\dfrac{1}{2} + \dfrac{1}{(j-i+1)(j-i)} + \dfrac{2(i-1)}{(j-i+2)(j-i+1)(j-i)}$	$\dfrac{n^2}{2}$
`QuickMin`	$\dfrac{2}{j}$	$2n$
`SelMin`	$\dfrac{1}{i(i+1)} + \dfrac{1}{j(j-1)}$	n

TABLE *1.1 – Nombre moyen de comparaisons entre les clefs de rangs i et j ($i < j$) dans le modèle uniforme des permutations.*

Considérons l'exemple extrêmement classique de l'algorithme Quicksort présenté dans la section 1.1.2 p. 4. Pour calculer $\pi_n(i,j)$, remarquons que tant que le pivot n'appartient pas au sous-ensemble $\mathcal{U}_{[i,j]}$, ce sous-ensemble est placé en entier à gauche ou à droite du pivot après partitionnement ce qui n'induit pas de comparaisons entre U_i et U_j. Lorsqu'un pivot appartient au sous-ensemble $\mathcal{U}_{[i,j]}$, les clefs U_i et U_j ne peuvent être comparées que si l'une des deux est le pivot. Cet événement coïncide avec l'événement « U_i ou U_j est choisi comme pivot, et ce pivot est le premier élément du sous-ensemble $\mathcal{U}_{[i,j]}$ à apparaître dans la séquence d'entrée ». Après une telle comparaison, les deux clefs sont séparées après partitionnement et ne seront plus jamais comparées par l'algorithme. La probabilité (par exemple) que la clef U_i soit la première dans le sous-ensemble $\mathcal{U}_{[i,j]}$ à apparaître dans l'entrée est exactement $1/\operatorname{Card}(\mathcal{U}_{[i,j]}) = 1/(j-i+1)$. C'est également en raisonnant sur les positions τ la probabilité de l'évenement « $(\forall k \in [i\,..\,j])\ \tau(U_i) \leq \tau(U_k)$ ». La situation est symétrique pour U_j et on en déduit que le nombre moyen de comparaisons entre les clefs U_i et U_j est $\pi_n(i,j) = 2/(j-i+1)$.

Par des arguments similaires (plus ou moins compliqués) on complète la deuxième colonne de la table 1.1 pour chacun des algorithmes.

1.5.2 La source génère les clefs

Clefs versus symboles

Nous sommes intéressés par un coût plus réaliste pour la comparaison de deux clefs que le simple coût unitaire usuel. On souhaite donc étudier en moyenne le nombre de comparaisons de symboles. Nous considérons ici le modèle où une source probabilisée $\mathcal{S}$ produit indépendamment n clefs en tirant n paramètres réels $(v_1, v_2, \ldots, v_n)$ uniformément de l'intervalle $[0, 1]$. Les clefs (i.e., mots infinis) correspondantes $(M(v_1), M(v_2), \ldots, M(v_n))$ sont donc obtenues via l'application M de la définition 1.3.6 décrite à la section 1.3.4. Rappelons que l'application M préserve l'ordre total existant sur l'intervalle $[0, 1]$ ($x > y$ si et seulement si $M(x) \succ M(y)$ pour $x, y \in [0, 1]$ et '$\succ$' désigne l'ordre lexicographique).

Les mots sont ordonnés grâce à l'ordre lexicographique, et le coût de la comparaison de deux clefs (le nombre de comparaisons de symboles nécessaires) est relié à la fonction de coïncidence définie comme suit.

Définition 1.5.1. *La* fonction de coïncidence $\gamma(u, t) : [0, 1] \times [0, 1] \to \mathbb{N} \cup \{+\infty\}$ *est la longueur du plus long préfixe commun entre les mots $M(u)$ et $M(t)$.*

Plus précisément, le coût réaliste de la comparaison entre $M(u)$ et $M(t)$ est égale à $\gamma(u, t) + 1$.

Nous représentons le couple $(M(u), M(t))$ avec $u \leq t$ par le point (u, t) du triangle $\mathcal{T} := \{(u, t) : 0 \leq u \leq t \leq 1\}$. Les *triangles fondamentaux*

$$\mathcal{T}_w = (\mathcal{I}_w \times \mathcal{I}_w) \cap \mathcal{T} = \{(u, t) : a_w \leq u \leq t \leq b_w\} \tag{1.23}$$

définissent les lignes de niveau de la fonction γ. En effet la coïncidence $\gamma(u, t)$ est au moins ℓ si et seulement si $M(u)$ et $M(t)$ ont le même préfixe commun w de longueur ℓ, si bien que les paramètres u et t appartiennent au même intervalle fondamental $\mathcal{I}_w$ relatif au préfixe w de longueur ℓ. De plus les deux relations

$$\mathcal{T} \cap [\gamma \geq \ell] = \bigcup_{w \in \Sigma^\ell} \mathcal{T}_w, \qquad \sum_{\ell \geq 0} \mathbf{1}_{[\gamma \geq \ell]} = \sum_{\ell \geq 0} (\ell + 1) \mathbf{1}_{[\gamma = \ell]},$$

entraînent l'égalité suivante vraie pour toute fonction g intégrable sur le triangle unité $\mathcal{T}$

$$\int_{\mathcal{T}} [\gamma(u, t) + 1] g(u, t)\, du\, dt = \sum_{w \in \Sigma^\star} \int_{\mathcal{T}_w} g(u, t)\, du\, dt. \tag{1.24}$$

La figure 1.3 suivante représente la famille de triangles $\mathcal{T}_w$, qui définit la « géométrie » de la source pour deux sources sans mémoire.

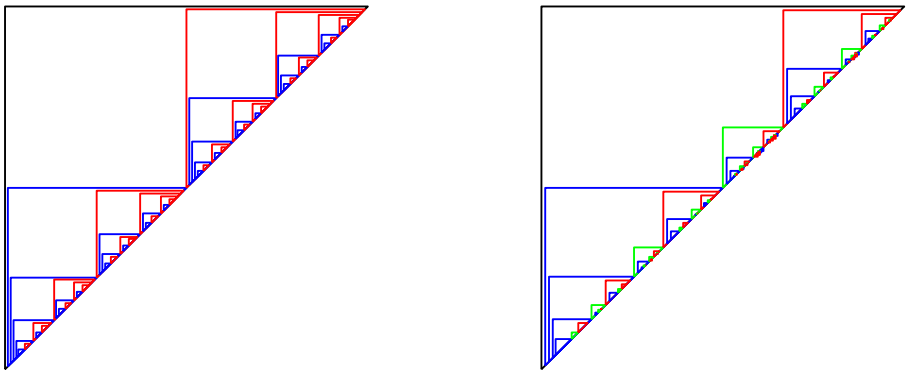


FIGURE 1.3 – *La géométrie de deux sources sans mémoire. À gauche, le cas binaire symétrique avec* $\Sigma := \{a, b\}$ *et* $p_a = p_b = 1/2$*. À droite, le cas d'un alphabet ternaire* $\Sigma := \{a, b, c\}$ *avec* $p_a = 1/2, p_b = 1/6, p_c = 1/3$*.*

Analyse en moyenne : quel modèle ?

Le but de l'analyse en moyenne d'algorithmes est de caractériser la valeur moyenne de leurs « coûts » sous un modèle probabiliste qui décrit la distribution initiale des entrées.

Le modèle que nous venons de présenter est strictement équivalent à un autre modèle. Ce modèle considère d'abord un ensemble *ordonné* de points $\{u_1 < u_2 < \cdots < u_n\}$ aléatoires indépendants avec la distribution uniforme sur $[0, 1]$ (avec probabilité 1 tous les points sont distincts). Cet ensemble a donc pour image par l'application M l'ensemble $\mathcal{U} = \{M(u_1) < \cdots < M(u_n)\}$, sur lequel on peut faire agir une permutation aléatoire uniforme $\sigma \in \mathfrak{S}_n$ pour constituer l'entrée.

Ainsi, pour un ensemble de clefs $\mathcal{U}$ fixé et un sous-intervalle $\mathcal{J} \subset [0, 1]$, nous notons $N_{\mathcal{J}} = \text{Card}(\{x \in \mathcal{J} \mid M(x) \in \mathcal{U}\})$ le nombre de clefs dont le paramètre appartient à l'intervalle $\mathcal{J}$. Avec cette notation, considérons d'abord le nombre moyen $\widetilde{\pi}(u, t)$ de comparaisons entre les clefs $M(u)$ et $M(t)$ effectuées par l'algorithme, où la moyenne est effectuée par rapport à toutes les permutations sur $\mathcal{U}$. Si u et t sont des paramètres de clefs appartenant à $\mathcal{U}$, le nombre moyen $\widetilde{\pi}(u, t)$ est lié au nombre moyen $\pi_n(i, j)$ par l'égalité

$$\widetilde{\pi}(u,t) = \pi_n(N_{[0,u[}+1, N_{[0,u[}+N_{]u,t[}+2), \text{ avec } n = N_{[0,1]} = N_{[0,u[} + N_{]u,t[} + N_{]t,1]} + 2. \tag{1.25}$$

En effet considérons un univers de clefs $\mathcal{U}$ contenant deux clefs $M(u)$ et $M(t)$. Cet ensemble est obtenu à partir de points de l'intervalle $[0, 1]$ par l'application M. Le nombre total de clefs est bien $N_{[0,1]}$ et les clefs $M(u)$ et $M(t)$ sont

de rangs respectifs $N_{[0,u[}+1$ et $N_{[0,u[}+N_{]u,t[}+2$, où les translations respectives de 1 et 2 reflètent que $M(u)$ et $M(t)$ appartiennent à $\mathcal{U}$.

Lorsque le cardinal n de l'ensemble des clefs $\mathcal{U}$ est fixé, et que ces mots $U_i \in \mathcal{U}$ sont émis indépendamment par la source $\mathcal{S}$, le modèle est dit *de Bernoulli* et noté $(\mathcal{B}_n, \mathcal{S})$. Néanmoins il est techniquement plus facile de considérer que l'entrée (l'ensemble $\mathcal{U}$) est de taille variable N et que cette taille suit une loi de Poisson de taux Z définie par

$$\Pr\{N = k\} = e^{-Z}\frac{Z^k}{k!}. \tag{1.26}$$

Dans ce modèle, appelé modèle de Poisson de taux Z, le paramètre Z joue une rôle très similaire[6] au cardinal n dans le modèle de Bernoulli $(\mathcal{B}_n, \mathcal{S})$. Quand les mots sont de plus produits par une source probabilisée $\mathcal{S}$, le modèle aléatoire désigné par $(\mathcal{P}_Z, \mathcal{S})$, s'envisage comme un processus en deux étapes :

(a) Le nombre N de mots est tiré aléatoirement selon la loi de Poisson de taux Z ;

(b) Ensuite, N mots sont produits indépendamment par la source $\mathcal{S}$.

Notons que dans le modèle de Poisson, les variables $N_{[0,u[}, N_{]u,t[}$ sont elles-mêmes des variables aléatoires de Poisson de taux respectifs Zu et $Z(t-u)$ qui ont le (bon) goût d'être indépendantes. *Cette propriété est d'ailleurs essentielle*[7]. Dans le modèle de Poisson, la variable $\widetilde{\pi}_(u,t)$ est donc elle-même une variable aléatoire fonction de ces variables.

Une formule exacte pour le nombre moyen de comparaisons de symboles

Nous travaillons d'abord dans le modèle de Poisson, où les propriétés d'indépendance permettent de simplifier les calculs, puis nous reviendrons au modèle de Bernoulli, plus naturel.

Modèle de Poisson. La densité d'un algorithme $\mathcal{A}$ au point (u,t) du triangle unité $\mathcal{T}$ est défini comme le nombre moyen de comparaisons de clefs entre deux clefs (mots) $M(u')$ et $M(t')$ pour $u' \in [u-du, u]$ et $t' \in [t, t+dt]$. Cette densité est relativement facile à calculer dans le modèle de Poisson, comme nous allons le voir. Le nombre de comparaisons de clefs effectuées par l'algorithme entre deux mots $M(u')$ et $M(t')$ pour $u' \in [u-du, u]$ et $t' \in [t, t+dt]$ est égal à

$$N_{[u-du,u]} \cdot N_{[t,t+dt]} \cdot \widetilde{\pi}(u,t). \tag{1.27}$$

6. De fait la moyenne du cardinal de $\mathcal{U}$ est exactement égale à Z.

7. Et pas trop dure à montrer.

Les trois variables aléatoires correspondent à trois intervalles disjoints $[u - du, u]$, $[t, t + dt]$ et $[u, t]$. Dans le modèle de Poisson, ces variables sont donc indépendantes et suivent une loi de Poisson. Ainsi le nombre moyen de comparaisons de clefs est

$$\mathbb{E}_Z[N_{[u-du,u]}] \cdot \mathbb{E}_Z[N_{[t,t+dt]}] \cdot \mathbb{E}_Z[\widetilde{\pi}(u,t)] = Zdu \cdot Zdt \cdot \mathbb{E}_Z[\widetilde{\pi}(u,t)].$$

Alors, la densité $\phi_Z(u,t)$ dans le modèle de Poisson satisfait

$$\phi_Z(u,t) = Z^2 \cdot \mathbb{E}_Z[\widetilde{\pi}(u,t)]. \tag{1.28}$$

Dans le modèle $(\mathcal{P}_Z, \mathcal{S})$, cette quantité permet de calculer, non seulement le nombre moyen de comparaisons de clefs C_Z effectuées par l'algorithme, mais aussi le nombre moyen de comparaisons de symboles S_Z (en prenant en compte la coïncidence) grâce aux formules

$$C_Z = \int_{\mathcal{T}} \phi_Z(u,t)\, du\, dt, \qquad S_Z = \int_{\mathcal{T}} [\gamma(u,t)+1]\phi_Z(u,t)\, du\, dt.$$

Modèle de Bernoulli. Nous pouvons facilement revenir au modèle de Bernoulli $(\mathcal{B}_n, \mathcal{S})$, pour retrouver le nombre moyen de comparaisons de clefs $C(n)$ et le nombre moyen de comparaisons de symboles $S(n)$. En effet, l'espérance dans le modèle de Poisson étant égale à

$$S_Z = e^{-Z} \sum_{n \geq 0} \frac{S(n)}{n!},$$

le principe dit de « dépoissonisation algébrique » se traduit par l'égalité obtenue par extraction de coefficients

$$S(n) = n![Z^n]e^Z S_Z.$$

Pour calculer les coefficients $S(n)$, nous allons donc calculer son analogue S_Z dans le modèle de Poisson car il s'avère beaucoup plus facile à obtenir.

Principes pour un transfert automatique.

Les expressions de $\pi_n(i,j)$ étant explicites, on pourrait maintenant pour chacun des algorithmes calculer l'expression de S_Z dans le modèle de Poisson grâce aux équations (1.25) et (1.28).

Nous choisissons un chemin alternatif mais plus systématique où on prend en compte la forme particulière des quantités $\pi_n(i,j)$ observés. Il est

tout à fait remarquable (et pas complètement élucidé) que, comme nous allons le voir, la variable aléatoire $\widetilde{\pi}(u,t)$ s'écrive comme une combinaison linéaire de fonctions rationnelles particulières dont les espérances de Poisson sont faciles à obtenir, et ce pour *tous* les algorithmes de notre étude.

Ensuite, dans le cadre de notre méthode, nous écrivons l'espérance de Poisson $\phi_Z(u,t)] = Z^2\mathbb{E}_Z[\widetilde{\pi}(u,t)]$ de l'équation (1.28) sous la forme d'une série alternée [8]

$$\phi_Z(u,t) = \sum_{k\geq 2}(-1)^k \frac{Z^k}{k!}\varphi(k,u,t). \tag{1.29}$$

Les coefficients $\varphi(k,u,t)$ sont donc définis par

$$\varphi(k,u,t) = (-1)^k k![Z^k]\phi_Z(u,t) \text{ pour } k\geq 2, \qquad \varphi(k,u,t) = 0 \text{ pour } k = 0,1. \tag{1.30}$$

L'idée d'écrire S_Z sous la forme (1.29) n'est pas (entièrement !) gratuite et permet de fait de systématiser les calculs et de simplifier la présentation de l'analyse asymptotique qui suivra.

Nous présentons brièvement une manière semi-automatique de calculer les coefficients $\varphi(k,u,t)$ de l'équation (1.33) à partir des quantités $\pi_n(i,j)$ pour donner lieu aux résultats rassemblés dans la table 1.2 pour les cinq algorithmes.

Proposition 1.5.2. *Soit une variable aléatoire X suivant une loi de Poisson de taux Z, et, pour $m\geq 1$, la variable*

$$f_m(X) := \frac{1}{(X+1)(X+2)\dots(X+m)}.$$

Alors on a

(a) *Pour chacun des algorithmes étudiés, la variable aléatoire $\widetilde{\pi}(u,t)$ s'exprime comme la somme de fonctions de la « base » (f_m) et d'une éventuelle constante (voir la deuxième colonne de la table 1.2).*

(b) *Soit $F_m(Z)$ la valeur moyenne de $f_m(X)$ dans le modèle de Poisson de taux Z. Alors les deux suites*

$$\beta_m(n,\lambda) = (-1)^n n![Z^n]\left(Z^2\, F_m(\lambda Z)\right)$$
$$\gamma_m(n,\lambda) = (-1)^n n![Z^n]\left(Z^3\, F_m(\lambda Z)\right).$$

8. Le signe $(-1)^k$ permet d'obtenir des coefficients $\varphi(k,u,t)$ de même signe et également de simplifier la présentation et les calculs de l'étude asymptotique qui reste à venir. Enfin la condition $k\geq 2$ vient de l'équation (1.28).

admettent les expressions suivantes[9] *respectivement pour* $n > 1$ *et* $n > 2$,

$$\beta_m(n,\lambda) = \tfrac{1}{(m-1)!}\tfrac{n(n-1)}{n+m-2}\lambda^{n-2}, \quad \gamma_m(n,\lambda) = \tfrac{-1}{(m-1)!}\tfrac{n(n-1)(n-2)}{n+m-3}\lambda^{n-3}. \tag{1.31}$$

(c) *Pour chacun des algorithmes il existe un entier* σ_0*, pour lequel les coefficients* $\varphi(n,u,t)$ *(calculés pour une densité* $\Phi_Z(u,t)$*) s'exprime pour* $n > \sigma_0$ *comme une combinaison linéaire de termes* $\beta_m(n,\lambda)$ *et* $\gamma_m(n,\lambda)$ *pour* $\lambda \in \{u, t, t-u\}$*, comme décrit dans la troisième colonne de la table 1.2. L'entier* σ_0 *est donné dans la quatrième (et dernière colonne) de la table 1.2.*

Algorithmes	$\widetilde{\pi}(u,t)$ (dans la « base » f_m)	$\varphi(n,u,t)$ (dans la « base » β_i, γ_j)	σ_0
`QuickSort`	$2\left(f_1(N_{[u,t[}) - f_2(N_{[u,t[})\right)$	$2\left(\beta_1(n,t-u) - \beta_2(n,t-u)\right)$	1
`InsSort`	$\frac{1}{2} + f_2(N_{[u,t[})$	$\beta_2(n,t-u)$	2
`BubSort`	$\frac{1}{2} + f_2(N_{[u,t[}) + 2N_{[0,u[} \cdot f_3(N_{[u,t[})$	$\beta_2(n,t-u) + 2u\gamma_3(n,t-u)$	2
`QuickMin`	$2\left(f_1(N_{[0,t[}) - f_2(N_{[0,t[})\right)$	$2\left(\beta_1(n,t) - \beta_2(s,t)\right)$	1
`SelMin`	$f_2(N_{[0,u[}) + f_2(N_{[0,t[})$	$\beta_2(n,u) + \beta_2(n,t)$	1

TABLE *1.2 – Transfert automatique de* $\pi_n(i,j)$ *vers* $\varphi(n,u,t)$ *valable pour* $n > \sigma_0$.

Démonstration.
Assertion (a). On part de l'expression de $\pi_n(i,j)$ dans la table 1.1, et on obtient avec (1.25) une expression pour la variable aléatoire $\widetilde{\pi}(u,t)$ correspondant à la seconde colonne de la table 1.2.

Assertion (b). Rappelons que

$$F_m(Z) := \mathbb{E}_Z[f_m(X)] = e^{-Z}\sum_{k\geq 0} f_m(k)\frac{Z^k}{k!}.$$

Nous calculons d'abord les coefficients $\alpha_m(n,\lambda)$ de $F_m(Z)$

$$\alpha_m(n) := (-1)^n n![Z^n]F_m(Z) = \frac{1}{(m-1)!}\frac{1}{n+m}. \tag{1.32}$$

Ensuite les coefficients $\beta_m(n,\lambda)$ et $\gamma_m(n,\lambda)$ sont reliés à $\alpha_m(n)$, respectivement pour $n > 1$ et $n > 2$

$$\beta_m(n,\lambda) = n(n-1)\,\lambda^{n-2}\alpha_m(n-2), \qquad \gamma_m(n,\lambda) = -n(n-1)(n-2)\,\lambda^{n-3}\alpha_m(n-3),$$

9. Notons que $\beta_m(n,\lambda) = 0$ pour $n \leq 1$ et $\gamma_m(n,\lambda) = 0$ pour $n \leq 2$.

ce qui prouve, avec (1.32), les expressions (1.31) de l'assertion (b).

Assertion (c). La troisième colonne est obtenue à partir de la seconde en prenant l'espérance dans le modèle de Poisson, en multipliant par Z^2 et en extrayant le coefficient en Z^n. Toutes les expressions de la seconde colonne sont des expressions linéaires, excepté le terme $N_{[0,u[} \cdot f_3(N_{]u,t[})$, qui est le produit de deux variables aléatoires indépendantes (dans le modèle de Poisson). L'espérance du produit est dans ce cas le produit des espérances

$$Z^2 \cdot \mathbb{E}_Z(N_{[0,u[}) \cdot \mathbb{E}_Z\left(f_3(N_{]u,t[})\right) = u \cdot Z^3 F_3(Z(t-u)).$$

Enfin, notons que l'on calcule explicitement la valeur de $\varphi(\sigma_0)$ dans le cas ou $\sigma_0 = 2$ (et qui sera nécéssaire pour le calcul plus tard) à partir de l'expression de $\pi_n(i,j)$. Cette valeur correspond au fait d'avoir un terme constant $\frac{1}{2}$ dans $\pi_n(i,j)$. Dans le cas des deux algorithmes concernés ici (`InsSort` et `BubSort`), on calcule facilement le premier terme non nul du développement limité de l'équation $\varphi(2,u,t) = 1$ dans l'équation (1.29), et par intégration on obtient que $\varphi(2) = \frac{\Lambda(2)}{2} = \frac{1}{2}\sum_{w\in\Sigma^*} p_w^2$. □

Les expressions explicites de $\varphi(n,u,t)$ découlent des décompositions de la troisième colonne de la table 1.2 et des expressions de l'équation (1.31). Utilisant l'équation (1.24), la suite $\varphi(k)$ est définie pour $k \geq 2$,

$$\varphi(k) := \int_{\mathcal{T}} (\gamma(u,t)+1)\,\varphi(k,u,t)\,du\,dt = \sum_{w\in\Sigma^\star} \int_{\mathcal{T}_w} \varphi(k,u,t)\,du\,dt, \qquad (1.33)$$

et s'obtient en calculant l'intégrale de $\varphi(k,u,t)$ sur les triangles fondamentaux $\mathcal{T}_w$. Les résultats de ces calculs sont rassemblés dans la table 1.3. Le terme $(-1)^k\varphi(k)$ est alors justement le coefficient du développement en série de S_Z,

$$S_Z = \sum_{k\geq 0} (-1)^k \frac{Z^k}{k!}\varphi(k). \qquad (1.34)$$

Le nombre moyen de comparaisons de symboles $S(n)$ effectuées par l'algorithme sur une entrée constituée de n mots produit indépendamment par la même source probabilisée est relié à S_Z et donc à $\varphi(n)$ par les égalités

$$S_Z = e^{-Z}\sum_{n\geq 2} \frac{Z^n}{n!} S(n), \qquad S(n) = \sum_{k=2}^{n} (-1)^k \binom{n}{k} \varphi(k). \qquad (1.35)$$

On obtient donc une expression exacte pour $S(n)$, qui plus est obtenue de manière « semi-automatique » à partir de l'expression de la valeur moyenne

Algorithmes	$\varphi(n,u,t)$ (pour $n > \sigma_0$)	$\varphi(n)$ (pour $n > \sigma_0$)	σ_0
`QuickSort`	$2(t-u)^{n-2}$	$\frac{2\Lambda(n)}{n(n-1)} = \frac{2}{n(n-1)} \sum_{w\in\Sigma^\star} p_w^n$	1
`InsSort`	$(n-1)(t-u)^{n-2}$	$\frac{\Lambda(n)}{n} = \frac{1}{n} \sum_{w\in\Sigma^\star} p_w^n$	2
`BubSort`	$(n-1)(t-u)^{n-3}[t-(n-1)u]$	$-\sum_{w\in\Sigma^\star} a_w p_w^{n-1}$	2
`QuickMin`	$2t^{n-2}$	$2\sum_{w\in\Sigma^\star} \int_{a_w}^{b_w} (t-a_w)t^{n-2}dt$	1
`SelMin`	$(n-1)[u^{n-2}+t^{n-2}]$	$(n-1)\sum_{w\in\Sigma^\star} (b_w-a_w)\int_{a_w}^{b_w} u^{n-2}du$	1

TABLE *1.3 – Calcul des coefficients $\varphi(n)$ pour les cinq algorithmes. Pour ces cinq algorithmes, on aura besoin également d'évaluer φ en σ_0 : si $\sigma_0 = 1$ on calcule $\varphi(1) = 0$ et si $\sigma_0 = 2$ on a $\varphi(2) = \frac{\Lambda(2)}{2}$.*

$\pi_n(i,j)$ (une cette quantité écrite dans une certaine « base » de fonctions $(f_m(X))$). En un sens, l'analyse est terminée ! Cependant le résultat obtenu est évidemment difficile à interpréter que ce soit pour l'ordre de grandeur ou évaluer l'impact du mécanisme qui a permis de produire les clefs (ce qui est quand même notre but). C'est pourquoi il reste à réaliser une étape décisive : évaluer le comportement asymptotique.

1.5.3 Analyse asymptotique pour le nombre moyen de comparaisons de symboles

L'expression (1.35) de $S(n)$ est très classiquement traitée à l'aide de la formule de Rice (donnée ci-dessous) afin d'obtenir un développement asymptotique de $S(n)$ lorsque $n \to \infty$. Cette étape est typique de la combinatoire analytique et met en jeu des fonctions de la variable complexe.

Formule de Rice. La formule de Rice [36, 37] transforme une somme binomiale en une intégrale dans le plan complexe. Pour un entier σ_0 et tout réel $\sigma_1 \in]\sigma_0, \sigma_0+1[$, on a

$$T(n) = \sum_{k=1+\sigma_0}^{n} (-1)^k \binom{n}{k} \varphi(k) \tag{1.36}$$

$$= \frac{(-1)^{n+1}}{2i\pi} \int_{\Re s=\sigma_1} G(s)\, ds, \text{ avec } G(s) = \tfrac{n!\,\varpi(s)}{s(s-1)\ldots(s-n)}, \tag{1.37}$$

où dans cette équation $\varpi(s)$ est un relèvement analytique de la séquence $\varphi(k)$ (i.e., $\varpi(k) = \varphi(k)$ pour $k \geq 1 + \sigma_0$).

Alors, selon un principe général en combinatoire analytique [24, 25], la droite d'intégration peut être déplacée vers la gauche, dès que $G(s)$ (et donc $\varpi(s)$) a de bonnes propriétés analytiques : Nous avons besoin d'une région $\mathcal{R}$ à gauche de la droite $\Re s = \sigma_0$, où $\varpi(s)$ soit de croissance polynomiale (pour $|\Im s| \to \infty$) et méromorphe. Avec une bonne connaissance des pôles de $G(s)$, nous obtenons finalement une formule de résidus

$$T(n) = (-1)^{n+1} \left[\sum_s \operatorname{Res}\left[G(s)\right] + \frac{1}{2i\pi} \int_{\Gamma_2} G(s)\, ds \right], \tag{1.38}$$

où Γ_2 est un chemin de classe $\mathcal{C}^1$ inclus dans la région $\mathcal{R}$ et la somme est effectuée sur tous les pôles s de $G(s)$ dans le domaine intérieur délimité par la droite verticale $\Re s = \sigma_1$ et la courbe Γ_2.

Les singularités dominantes de $G(s)$ donnent le comportement asymptotique de $T(n)$, et l'intégrale de reste (qui s'avère négligeable) est estimée en utilisant la croissance polynomiale de $G(s)$ lorsque $|\Im(s)| \to \infty$.

Relèvement analytique. Afin d'appliquer la formule de Rice, nous avons besoin de considérer un relèvement analytique de la suite $(\varphi(k))$ (ou encore un relèvement analytique $\varpi(s, u, t)$ pour les coefficients $\varphi(k, u, t)$), c'est-à-dire une fonction analytique $\varpi(s)$ qui coïncide avec $\varphi(k)$ aux valeurs entières $s = k$ dans la somme (1.34).

En règle générale, il n'est pas aisé de trouver un prolongement analytique [10] d'une suite quelconque. Cependant, ici, un simple changement de variable $n \to s$ dans la table 1.3 fournit le relèvement analytique $\varpi(s)$ de $(\varphi(k))_{k>\sigma_0}$ que l'on retrouve dans la table 1.4.

Un peu de soin est parfois nécessaire pour tenir compte la situation où le relèvement analytique ne coïnciderait sur quelques valeurs. Pour chaque algorithme, l'existence d'un relèvement n'est garanti que sur un certain domaine $\Re s > \sigma_0$. Cependant la valeur de σ_0 dépend de l'algorithme, et nous distinguons deux classes d'algorithmes. La première classe est constituée de `InsSort` et `BubSort`. Pour ces algorithmes un terme constant $1/2$ apparaît dans la valeur moyenne $\pi_n(i, j)$ (comme on le voit dans la table 1.1). Dans ce cas le relèvement analytique $\varpi(s)$ ne coïncide que pour les entiers $k \geq 3$. Le relèvement analytique n'est donc défini que pour $\Re s > 2$, et l'abscisse σ_0 est

10. Il peut y en avoir plusieurs par exemple, ce qui de manière surprenante de premier abord, ne change rien à la validité de la formule de Rice.

égale à 2. La deuxième classe contient les autres algorithmes pour lesquels le relèvement analytique coïncide pour $k \geq 2$, et existe pour $\Re s > 1$, et l'abscisse σ_0 vaut 1. D'après les équations (1.35) et (1.36), et dans le cas où $\sigma_0 = 2$, nous devons ajouter à $T(n)$ un terme correspondant à l'indice $k = 2$ afin d'obtenir la somme $S(n)$ qui nous intéresse réellement. Pour les algorithmes `BubSort` et `InsSort`, les termes additionnels sont de la forme $\varphi(2)\binom{n}{2}$. En conséquence, selon la valeur de σ_0, le nombre moyen de comparaisons de symboles est

$$S(n) = T(n) = \sum_{k=2}^{n} (-1)^k \binom{n}{k} \varpi(k) \qquad \text{(si } \sigma_0 = 1\text{)} \tag{1.39}$$

$$S(n) = \binom{n}{2}\varphi(2) + T(n) = \binom{n}{2}\varphi(2) + \sum_{k=3}^{n} (-1)^k \binom{n}{k} \varpi(k) \quad \text{(si } \sigma_0 = 2\text{)}, \tag{1.40}$$

où les expressions de $\varpi(s)$ sont données dans la deuxième colonne de la table 1.4 pour les cinq algorithmes de l'étude. Rappelons que dans notre étude, le cas $\sigma_0 = 2$ correspond aux algorithmes `InsSort` et `BubSort` et que l'on calcule $\varphi(2) = \frac{\Lambda(2)}{2}$.

La série obtenue $\varpi(s)$ est appelée série de Dirichlet *mixte* puisqu'elle mélange des propriétés de la source (via les triangles fondamentaux $\mathcal{T}_w$) et de l'algorithme ((via $\varpi(s, u, t)$). On peut écrire

$$\varpi(s) = \int_{\mathcal{T}} [\gamma(u,t) + 1] \varpi(s,u,t)\, du\, dt = \sum_{w \in \Sigma^\star} \int_{\mathcal{T}_w} \varpi(s,u,t)\, du\, dt. \tag{1.41}$$

Propriétés sur la série mixte de Dirichlet $\varpi(s)$.

Il ne reste maintenant qu'une seule chose à faire pour conclure à partir des équations (1.39),(1.39) et la formule de rice (1.36) : faire un calcul de résidu et donc examiner les singularités de $\varpi(s)$.

Jusqu'ici la source a été supposée relativement générale et on a aucune garantie sur le comportement analytique sur la série $\varpi(s)$ elle-même reliée d'assez prêt aux propriétés de $\Lambda(s)$.

Pour les sources usuelles (sans mémoire, Markov et sources dynamiques) il existe trois types principaux de régions $\mathcal{R}$ où de bonnes propriétés de $\varpi(s)$ rendent possible l'application de la formule de Rice et le fait de pouvoir décaler la droite verticale d'intégration vers la gauche. Plus précisément, on a besoin, par exemple pour l'application de la formule de Rice, d'une région

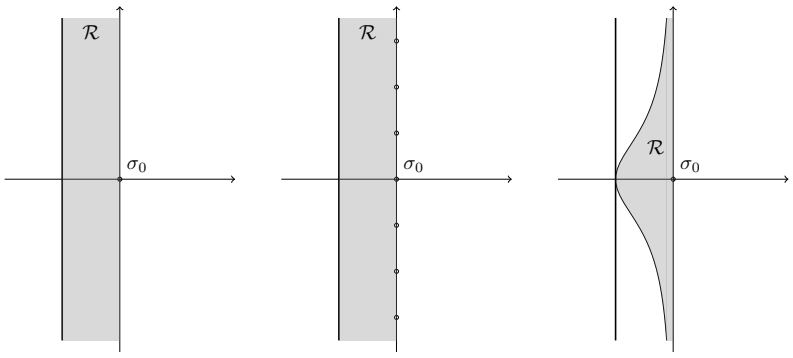


FIGURE 1.4 – *Les différentes régions sans pôles du plan complexe pour définir différents types de domestication. De gauche à droite : bande sans pôles, bande sans pôles avec des pôles périodiquement disposés sur une droite verticale, région sans pôle de forme hyperbolique.*

où les pôles de $\varpi(s)$ sont bien identifiés : un pôle d'un certain ordre k en une certaine abcisse qui sera ici l'abscisse σ_0, d'éventuels autres pôles mais qui laissent de « l'espace pour respirer » au pôle dominant (soit des pôles périodiquement espacés sur la droite d'absisse σ_0 mais garantissant une bande sans pôles, soit une zone hyperbolique sans pôles), enfin la fonction $\varpi(s)$ doit être d croissance polynomiale pour $|\Im s \to \infty|$ lorsque l'on s'éloigne de l'axe réel.

On dit alors que la fonction $\varpi(s)$ est « domestiquée »[11] sur une région $\mathcal{R}$ d'une certaine forme. Voir la figure 1.4 pour une représentation schématique des ces situations : « bande sans pôles », « bande sans pôles avec pôles disposés périodiquement sur une droite verticale » et « zone sans pôle de forme hyperbolique ». Pour plus de détails, nous renvoyons à [50] et [11]. Remarquons que d'après l'équation (1.38), le terme dominant de l'asymptotique provient des pôles dans la région $\mathcal{R}$ (par un calcul de résidus) et que le terme d'erreur est lui induit par l'évaluation de l'intégrale le long d'un chemin d'un chemin Γ_2 à l'intérieur de $\mathcal{R}$ dans l'équation (1.38).

Les séries mixtes de Dirichlet $\varpi(s)$ obtenues pour chacun des cinq algorithmes de cette étude sont fortement liées à la série de Dirichlet $\Lambda(s)$ de la source elle-même. Cette série a été définie dans la section 1.3.5 dans l'équa-

11. Selon les auteurs, on parle aussi de série disciplinée ou apprivoisée. Ces termes signifient simplement que la série admet un domaine où ses pôles et son comportement analytique sont bien « cadrés ».

tion (1.11)

$$\Lambda(s) = \sum_{w \in \Sigma^*} p_w^s, \qquad \Lambda_k(s) = \sum_{w \in \Sigma^k} p_w^s.$$

Il est essentiel pour avoir un résultat assez général de comprendre comment la série de Dirichlet de la source permet de garantir que la série mixte $\varpi(s)$ sera bien domestiquée. On introduit donc différentes notions (plus ou moins fortes) de domestication pour une source probabilisée. Pour plus de détails, nous engageons le lecteur à consulter les articles [23, 46, 39, 8, 11, 10]. Cette notion de domestication pour les sources est adaptée à notre cadre puisqu'il correspond à des situations où la série mixte $\varpi(s)$ peut être prouvée domestiquée.

Nous aurons essentiellement besoin de la propriété suivante. Par *définition* une source domestiquée possède une série de Dirichlet qui admet un développement [12] près de sa singularité $\sigma_0 = 1$

$$\Lambda(s) = \frac{1}{h(\mathcal{S})} \frac{1}{s-1} + O(1),$$

où $h(\mathcal{S})$ est l'entropie de la source (définie dans l'équation (1.12)).

Pour conclure et donc appliquer la formule de Rice (1.36), nous avons besoin de calculer le résidu de $G(s)$ en σ_0. Ceci revient principalement à connaître le développement de $\varpi(s)/(s-\sigma_0)$ au voisinage du pôle $s = \sigma_0$. Une fois ce développement obtenu, cela est chose facile (du moins tout logiciel de calcul formel en est capable) de calculer le résidu. *Notons que étudier les singularités de $\varpi(s)$ à partir de celles de $\Lambda(s)$ est l'une des difficultés de cette analyse.* Cela conduira au résultat final pour l'asymptotique de $S(n)$ du théorème 1.5.4.

Pour préciser un peu les choses et toujours avec l'exemple de `Quicksort`, on calcule aussi le développement pour la série mixte au voisinage de $s = 1$

$$\varpi(s) = \frac{2\Lambda(s)}{s(s-1)} = \frac{2}{h(\mathcal{S})} \frac{1}{(s-1)^2} + O(\frac{1}{s-1}).$$

Pour les algorithmes de tri, et dans les trois cas, les séries mixtes de Dirichlet sont étroitement liées à la série $\Lambda(s)$ de la source et le transfert de domestication entre $\Lambda(s)$ et $\varpi(s)$ est facile.

Dans le cas des algorithmes de sélection, l'étude analytique de $\varpi(s)$ est plus difficile, car $\varpi(s)$ dépend non seulement des probabilités p_w, mais aussi

12. Bien sûr on peut être plus précis si l'on souhaite évaluer les termes d'erreur ou les termes sous-dominants du développement asymptotique du nombre de comparaisons $S(n)$.

de l'intégrale d'une fonction sur chaque intervalle fondamental $[a_w, b_w]$. Dans ces deux cas on est amenés à considérer les séries mixtes par niveau $\varpi_\ell(s)$ de profondeur ℓ, correspondant aux intervalles fondamentaux de profondeur ℓ, et nous prouvons que la série de terme général $\varpi_\ell(s)$ converge normalement.

Les preuves ne sont pas données ici (on se référera à [11]).

Algorithmes	$\varpi(s)$	σ_0	Terme principal de $\varpi(s)/(s-\sigma_0)$
`QuickSort`	$\dfrac{2\Lambda(s)}{s(s-1)} = \dfrac{2}{s(s-1)} \displaystyle\sum_{w\in\Sigma^\star} p_w^s$	1	$\dfrac{2}{h(\mathcal{S})}\dfrac{1}{(s-1)^3}$
`InsSort`	$\dfrac{\Lambda(s)}{s} = \dfrac{1}{s} \displaystyle\sum_{w\in\Sigma^\star} p_w^s$	2	$\dfrac{c(\mathcal{S})}{2}\dfrac{1}{(s-2)}$
`BubSort`	$-\displaystyle\sum_{w\in\Sigma^\star} a_w p_w^{s-1}$	2	$-\dfrac{1}{2h(\mathcal{S})}\dfrac{1}{(s-2)^2}$
`QuickMin`	$2\displaystyle\sum_{w\in\Sigma^\star}\int_{a_w}^{b_w}(t-a_w)t^{s-2}dt$	1	$2b(\mathcal{S})\dfrac{1}{s-1}$
`SelMin`	$(s-1)\displaystyle\sum_{w\in\Sigma^\star}(b_w-a_w)\int_{a_w}^{b_w}u^{s-2}du$	1	$a(\mathcal{S})\dfrac{1}{s-1}$

TABLE *1.4 – Résultats pour l'étape 3 (étude de la singularité dominante de $\varpi(s)$ en $s=\sigma_0$).*

Principales constantes.

Nous décrivons ici les principales constantes qui interviennent devant le terme dominant de l'expression singulière de $\varpi(s)/(s-\sigma_0)$ en $s=\sigma_0$.

Proposition 1.5.3. *Les constantes qui interviennent dans la dernière colonne de la table 1.4 sont*

- (i) *l'entropie $h(\mathcal{S})$ de la source.*
- (ii) *La coïncidence $c(\mathcal{S})$, i.e., le nombre moyen de symboles nécessaires pour comparer deux mots aléatoires produits par la source (en examinant leurs préfixes).*
- (iii) *la min–coïncidence $a(\mathcal{S})$: C'est le nombre moyen de symboles nécessaires pour distinguer un mot aléatoire du mot le plus petit pouvant être émis par la source.*
- (iv) *La coïncidence logarithmique $b(\mathcal{S})$: c'est le nombre moyen de symboles nécessaires pour comparer deux mots X et Y tirés aléatoirement de la manière suivante : le mot X est produit aléatoirement et uniformément par la*

source pour un paramètre t, et Y est tiré aléatoirement avec $Y \geq X$, selon une densité $1/t$.

L'entropie $h(\mathcal{S})$ est définie dans l'équation (1.12). Les constantes $a(\mathcal{S})$, $b(\mathcal{S})$, $c(\mathcal{S})$ satisfont les inégalités $a(\mathcal{S}) < b(\mathcal{S}), c(\mathcal{S}) < 2b(\mathcal{S})$ et sont définies par

$$a(\mathcal{S}) = \sum_{\ell \geq 0} q_\ell, \quad b(\mathcal{S}) = \sum_{w \in \Sigma^\star} \int_{\mathcal{T}_w} \frac{1}{t}\, du\, dt, \quad c(\mathcal{S}) = \sum_{w \in \Sigma^\star} p_w^2 = \Lambda(2).$$

Ici q_ℓ est la probabilité qu'un mot préfixe de longueur ℓ commun avec le plus petit mot pouvant être émis par la source, $\mathcal{T}_w$ est le triangle fondamental défini dans (1.23) et $\Lambda(s)$ est défini dans (1.11).

Les constantes $a(\mathcal{S})$, $c(\mathcal{S})$ et $h(\mathcal{S})$ sont faciles à calculer pour une source sans mémoire. Pour la source sans mémoire $\mathcal{M}_r$ non biaisée avec un alphabet de taille r on a

$$a(\mathcal{M}_r) = c(\mathcal{M}_r) = \frac{r}{r-1}, \qquad h(\mathcal{M}_r) = \log r.$$

Pour la source binaire classique $\mathcal{B}_p$ sur l'alphabet $\{0, 1\}$, avec $p_0 = p$ et $p_1 = 1 - p$, on a

$$a(\mathcal{B}_p) = \frac{1}{1-p}, \quad c(\mathcal{B}_p) = \frac{1}{2p(1-p)}, \qquad h(\mathcal{B}_p) = -p \log p - (1-p)\log(1-p).$$

La constante $b(\mathcal{S})$ est plus difficile à calculer même dans le cas sans mémoire. Mais pour la source $\mathcal{M}_r$, on a (voir aussi [28, 20])

$$b(\mathcal{M}_r) = \sum_{\ell \geq 0} \Big(1 + \frac{1}{r^\ell} \sum_{k=1}^{r^\ell - 1} \log \frac{k}{r^\ell}\Big), \qquad b(\mathcal{M}_2) \doteq 2.639689120.$$

1.5.4 Résultat final

Avec les résultats des précédentes sections, nous obtenons le comportement asymptotique du nombre moyen $S(n)$ de comparaisons de symboles pour chacun des cinq algorithmes de cette étude. Il faut pour cela partir des expressions des séries mixtes de Dirichlet $\varpi(s)$ de la table 1.4 et calculer les résidus afin d'appliquer la formule de Rice de la section 1.5.3. Cela donne le théorème suivant.

Théorème 1.5.4. *Soit une source probabilisée $\mathcal{S}$ domestiquée. Pour les algorithmes* `QuickMin` *et* `SelMin`*, nous supposons que la source est faiblement domestiquée, et pour les algorithmes de tri* `QuickSort`, `InsSort`, `BubSort`*. Alors le nombre moyen $S(n)$ de comparaisons de symboles effectuées par chaque algorithme sur une entrée de n mots produit aléatoirement et indépendamment par la même source $\mathcal{S}$ admet le comportement asymptotique décrit dans la table 1.5. Les constantes dans les termes dominants ont déjà été décrites dans la proposition 1.5.3.*

Remarques. Les termes sous-dominants sont également accessibles ainsi que les termes d'erreurs, mais il faut être plus précis sur la domestication des séries (voir [11] pour des résultats plus avancés).

Algorithmes	$K(n)$	Terme dominant def $S(n)$
`QuickSort`	$2n \log n$	$\frac{1}{h(S)}\, n \log^2 n$
`InsSort`	$\frac{n^2}{4}$	$c(\mathcal{S}) 4\, n^2$
`BubSort`	$\frac{n^2}{2}$	$\frac{1}{4h(S)}\, n^2 \log n$
`QuickMin`	$2n$	$2b(\mathcal{S})\, n$
`SelMin`	n	$a(\mathcal{S})\, n$

TABLE *1.5 – Résultats du théorème 1.5.4.*

1.5.5 Discussion.

Nous pouvons comparer les comportements asymptotiques pour les deux valeurs moyennes, nombre moyen $C(n)$ de comparaisons de clefs (colonne 2 de la table 1.5) et nombre moyen de comparaisons de symboles $S(n)$ (colonne 3 de la table 1.5). On distingue ici deux types d'algorithmes :

(a) Les algorithmes « robustes » pour lesquelles $C(n)$ et $S(n)$ conservent le même ordre de grandeur. C'est le cas de trois algorithmes : `InsSort`, `QuickMin` et `SelMin`. Bien sûr, les constantes sont différentes pour $C(n)$ et $S(n)$, et les rapports $S(n)/C(n)$ font intervenir différents types de coïncidence, toujours entre *deux* mots, respectivement la coïncidence uniforme $c(\mathcal{S})$, la coïncidence logarithmique $b(\mathcal{S})$, ou la min-coïncidence $a(\mathcal{S})$ (décrites dans la section 1.5.3).

(b) Les algorithmes pour lesquelles $S(n)$ et $C(n)$ ne sont pas du même ordre, ici `QuickSort` et `BubSort`. Dans les deux cas le rapport $S(n)/C(n)$ satisfait [13]

$$\frac{S(n)}{K(n)} \sim \frac{1}{2h(\mathcal{S})} \log n. \tag{1.42}$$

Dans un autre registre, mentionnons les travaux dûs à Seidel [45] qui introduit la notion d'algorithme *fidèle*. Pour de tels algorithmes, on peut relier nombre moyen de comparaisons de clefs et nombre moyen de comparaisons de symboles. Nous pouvons adapter ces idées à notre cadre et en déduire un joli résultat de borne inférieure sur le nombre moyen de comparaisons de symboles pour tout algorithme de tri (reposant sur la procédure standard en informatique de comparaison entre deux chaînes de caractères). On peut ainsi montrer que la borne inférieure pour le nombre de comparaisons de symboles $\underline{S}(n)$ des algorithmes de tri basés sur la comparaison de clefs est

$$\underline{S}(n) \sim \frac{1}{2h(\mathcal{S})} n \log^2 n.$$

Ce résultat concerne uniquement les sources $\mathcal{S}$ « bien » domestiquées (que nous n'avons pas détaillées dans ce cours) mais qui restent assez générales puisqu'elles englobent notamment la plupart des sources usuelles comme la source sans mémoire et la source markovienne (pourvu qu'on ne choisisse pas des paramètres induisant un comportement pathologique – chaîne de Markov non primitive par exemple).

13. Ce qu'on ne sait pas encore totalement justifier a priori.

Bibliographie

[1] A. Akhavi and B. Vallée. Average bit-complexity of euclidean algorithms. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, ICALP '00, pages 373–387, London, UK, UK, 2000. Springer-Verlag.

[2] V. Baladi and B. Vallée. Euclidean algorithms are Gaussian. *Journal of Number Theory*, 110 :331–386, 2006.

[3] V. Berthé, J. Creusefond, L. Lhote, and B. Vallée. Multiple gcds. probabilistic analysis of the plain algorithm. In *Proceedings of the 38th International Symposium on International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, pages 37–44, New York, NY, USA, 2013. ACM.

[4] V. Berthé and H. Nakada. On continued fraction expansions in positive characteristic : Equivalence relations and some metric properties. *Expositiones Mathematicae*, 18 :257–284, 2000.

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

[6] J. Bourdon and B. Vallée. Pattern matching statistics on correlated sources. In *Proc. of LATIN'06*, volume 3887 of *LNCS*, pages 224–237, 2006.

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

[8] E. Cesaratto and B. Vallée. Gaussian distribution of trie depth for dynamical sources. submitted, 2012.

[9] B. Chauvin, J. Clément, and D. Gardy. Arbres et algorithmique. (titre provisoire, à paraître en 2014).

[10] J. Clément, T. Nguyen Thi, and B. Vallée. A general framework for the realistic analysis of sorting and searching algorithms. application to some popular algorithms. In *STACS*, pages 598–609, 2013.

[11] J. Clément, T. Nguyen Thi, and B. Vallée. Realistic analysis of sorting and searching algorithms : Design of a general framework and application to some popular algorithms. Special issue dedicated to the memory of Philippe Flajolet. Accepted, to appear., 2014.

[12] H. Daudé, P. Flajolet, and B. Vallée. An average-case analysis of the Gaussian Algorithm for Lattice Reduction. *Combinatorics, Probability & Computing*, 6(4) :397–433, 1997.

[13] H. Delange. Généralisation du théorème d'Ikehara. *Ann. Sc. ENS*, 71 :213–422, 1954.

[14] J. D. Dixon. The number of steps in the euclidean algorithm. *Journal of Number Theory*, 2(4) :414 – 422, 1970.

[15] D. Dolgopyat. On decay of correlations in Anosov flows. *Ann. of Math.*, 147(2) :357–390, 1998.

[16] H. Edwards. *Riemann's Zeta Function*. Dover Publications, 2001.

[17] W. Ellison and F. Ellison. *Prime numbers*. A Wiley-interscience publication. Wiley, 1985.

[18] Euclide. *Les éléments*. -300av J.C.

[19] J. A. Fill. Distributional convergence for the number of symbol comparisons used by Quicksort. Ann. Appl. Probab. (2012), to appear.

[20] J. A. Fill and S. Janson. The number of bit comparisons used by quicksort : an average-case analysis. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 300–307, 2004. Long version *Electron. J. Probab.* 17, Article 43, 1-22 (2012).

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

[22] J. A. Fill and T. Nakama. Distributional convergence for the number of symbol comparisons used by Quickselect. *CoRR*, abs/1202.2599, 2012. submitted.

[23] P. Flajolet, M. Roux, and B. Vallée. Digital trees and memoryless sources : from arithmetics to analysis. *Proceedings of AofA'10, DMTCS, proc AM*, pages 231–258, 2010.

[24] P. Flajolet and R. Sedgewick. Mellin transforms and asymptotics : Finite differences and Rice's integrals. *Theor. Comput. Sci.*, 144(1&2) :101–124, 1995.

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

[26] P. Flajolet and B. Vallée. Continued fraction algorithms, functional operators, and structure constants. *Theor. Comput. Sci.*, 194(1-2) :1–34, 1998.

[27] C. Friesen and D. Hensley. The statistics of continued fractions for polynomials over a finite field. *Proc. Amer. Math. Soc*, 124 :2661–2673, 1996.

[28] P. J. Grabner and H. Prodinger. On a constant arising in the analysis of bit comparisons in Quickselect. *Quaestiones Mathematicae*, 31(4) :303–306, 2008.

[29] H. Heilbronn. On the average length of a class of finite continued fractions. In P. Turán, editor, *Number Theory and Analysis*, pages 87–96. Springer US, 1969.

[30] D. Hensley. The number of steps in the euclidean algorithm. *Journal of Number Theory*, 49(2) :142 – 182, 1994.

[31] A. Knopfmacher and J. Knopfmacher. The exact length of the Euclidean algorithm in $F_q[X]$. *Mathematika*, 35 :297–304, 1988.

[32] D. E. Knuth. *The Art of Computer Programming*, volume 3 : Sorting and Searching. Addison-Wesley, third edition, 1998.

[33] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998.

[34] L. Lhote and B. Vallée. Gaussian laws for the main parameters of the Euclid algorithms. *Algorithmica*, 50(4) :497–554, 2008.

[35] K. Ma and J. von zur Gathen. Analysis of Euclidean algorithms for polynomials over finite fields. *Journal of Symbolic Computation*, 9(4) :429 – 455, 1990.

[36] N. E. Nörlund. Leçons sur les équations linéaires aux différences finies. In *Collection de monographies sur la théorie des fonctions*. Gauthier-Villars, Paris, 1929.

[37] N. E. Nörlund. *Vorlesungen über Differenzenrechnung*. Chelsea Publishing Company, New York, 1954.

[38] R. F. Rice. Some practical universal noiseless coding techniques. Technical Report JPL-79-22, JPL, Pasadena, CA, 1979.

[39] M. Roux and B. Vallée. Information theory : Sources, dirichlet series, and realistic analyses of data structures. In *Proceedings 8th International Conference Words 2011*, volume 63 of *EPTCS*, pages 199–214, 2011.

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

[41] D. Salomon. *Data Compression : The Complete Reference*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2007. With contributions by Giovanni Motta and David Bryant.

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

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

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

[45] R. Seidel. Data-specific analysis of string sorting. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1278–1286, 2010.

[46] B. Vallée. Dynamical sources in information theory : Fundamental intervals and word prefixes. *Algorithmica*, 29(1/2) :262–306, 2001.

[47] B. Vallée. Dynamical analysis of a class of Euclidean algorithms. *Theoretical Computer Science*, 297(1–3) :447–486, Mar. 2003.

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

[49] B. Vallée, J. Clément, J. A. Fill, and P. Flajolet. The number of symbol comparisons in QuickSort and QuickSelect. In S. A. *et al.*, editor, *Proceedings of ICALP 2009, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 750–763. Springer-Verlag, 2009.

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

[51] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.