Combinatoire analytique et algorithmique des ensembles de données.

Marianne Durand

Ecole Doctorale de l'Ecole Polytechnique

Combinatoire analytique et algorithmique des ensembles de données

THÈSE

présentée et soutenue publiquement le 30 Avril 2004

pour l'obtention du

Doctorat de l'Ecole Polytechnique

(Spécialité : Mathématiques et Informatique)

par

Marianne Durand

Composition du jury

Rapporteurs :	Brigitte Chauvin
	Michael Drmota
Examinateurs :	Philippe Flajolet (directeur)
	Nicolas Pouyanne
	Jean-Marc Steyaert
	Brigitte Vallée
	Jean Vuillemin

Projet Algo, INRIA Rocquencourt

Remerciements

Je remercie mon directeur, Philippe Flajolet, pour son encadrement chaleureux et enthousiasmant, ainsi que pour tout le temps qu'il m'a consacré.

Brigitte Chauvin et Michael Drmota ont accepté immédiatement d'être mes rapporteurs, et je les remercie de leur lecture et de leurs commentaires.

Je remercie vivement Nicolas Pouyanne, Jean-Marc Steyaert, Brigitte Vallée et Jean Vuillemin qui me font l'honneur d'être dans mon jury.

J'ai eu le plaisir de travailler avec Hervé Brönnimann et Frédéric Cazals, je les remercie de leur gentillesse et de leur confiance. Je souhaite également remercier Alfredo "Tuba" Viola pour son accueil extrêmement chaleureux et ses nombreux conseils.

Je souhaite ici remercier les membres du projet Algo, avec qui j'ai eu le plaisir de travailler pendant ces années de thèse. Je remercie en particulier Frédéric Chyzak et Ludovic Meunier, mes co-bureaux successifs, pour toutes les discussions scientifiques ou non que j'ai pu avoir avec eux. Je souhaite également remercier Bruno Salvy et Philippe Dumas pour leur disponibilité, ainsi que Virginie Collette pour son efficacité inégalable. Je tiens à remercier Julien Fayolle, Frédéric Giroire, Marni Mishna, Pierre Nicodème, Mireille Régnier, Vincent Puyhaubert, Alexandre Sedoglavic et Mathias Vandenbogaert qui rendent le projet Algo accueillant et sympathique.

J'ai bénéficié de l'accueil des communautés Alea et AofA. Les conférences auxquelles j'ai pu assister m'ont permis de découvrir des problématiques variées et souvent passionnantes.

Je remercie Matthieu Finiasz et Vincent Simonet pour leurs précieux conseils.

François Maurel m'a soutenue et accompagnée pendant cette thèse en sachant m'encourager dans les moments de succès et me redonner confiance en moi pendant les moments de doute. Je suis heureuse de pouvoir l'en remercier aujourd'hui.

Résumé

Cette thèse traite d'algorithmique des ensembles de données en adoptant le point de vue de la combinatoire analytique. Cette approche permet d'obtenir des résultats asymptotiques puissants sur des questions variées. On traite ici de trois problèmes qui illustrent cette approche : les listes à sauts associées à de l'analyse asymptotique bivariée, le hachage à essai aléatoire avec pagination et le comptage probabiliste.

Les listes à sauts sont une structure de données intermédiaire entre les *skiplists* et les arbres binaires de recherche. Il s'agit d'une liste chaînée, où chaque nœud possède en plus un pointeur "jump" qui permet de se déplacer rapidement dans la liste. L'étude de cette structure a donné lieu à un problème d'asymptotique bivariée avec coalescence de singularités, qui est l'extraction du coefficient $[z^n u^k](1-z)^a(1-u)^b(1-zu)^c$.

Le hachage avec essai aléatoire est un algorithme qui gère les collisions d'une table de hachage. A chaque élément distinct est associée une séquence aléatoire de positions du tableau, et un élément est inséré dans la première case non pleine indiquée par cette séquence. Dans le contexte étudié qui est celui de la pagination, on obtient la moyenne, ainsi que tous les moments successifs du coût de construction. Dans le cas particulier où la table est pleine, on retrouve les résultats du problème du collectionneur de coupons généralisé.

La dernière partie est consacrée à la question très simple : Étant donné un ensemble de données de taille quelconque, comment estimer avec précision le nombre d'éléments distincts dans cet ensemble, en utilisant une mémoire auxiliaire très petite ? Les algorithmes originaux Loglog et Super Loglog permettent d'estimer le cardinal d'un ensemble en utilisant un kilooctet de mémoire avec une précision d'environ 3%.

Abstract

This thesis is about algorithms on data sets, from the point of view of analytic combinatorics. This approach leads to powerful results on various open problems. Its content is based on three main directions : jumplists and bivariate asymptotic, random probing hashing and probabilistic counting.

A jumplist is a data structure inspired from skiplists, and close to binary search trees. It is a sorted linked list, where each node has an additional pointer "jump" that allows to move quickly in the list. The study of this structure leads to a bivariate asymptotic problem, with coalescing singularities, that is the extraction of the coefficient $[z^n u^k](1-z)^a(1-u)^b(1-zu)^c$.

Random probing hashing is an algorithm that handles the problem of collisions in a hash table. To every distinct element is associated a random sequence of locations, and an element is inserted in the first non-full location of its sequence. In the context studied here, that is hashing with buckets, we find the average, and all the other moments of the construction cost of the table. In the particular case where the table is full, we rediscover the results of the generalized coupon collector.

The last part is concerned with the very simple question : Given a data set, how to estimate with accuracy the number of distinct elements, using a very small auxiliary memory ? The original algorithms Loglog and super Loglog tell how to estimate the cardinality of a data set using a kilobyte of memory, with a precision of about 3%.

Table des matières

Introduction

Le domaine étudié dans cette thèse est *l'algorithmique des ensembles de données*. On s'intéresse à des algorithmes permettant de stocker des données, de les manipuler (insertion, suppression, recherche, tri) et d'en extraire des informations qualitatives. Les méthodes utilisées pour *analyser* ces algorithmes sont l'emploi de séries génératrices, et sur ces séries, l'utilisation de théorèmes d'analyse complexes univariée ou bivariée comme l'analyse de singularités, la transformation de Mellin ainsi que la méthode du col.

Cette thèse présente donc deux aspects, qui sont liés. Le premier est un aspect *algorithmique pratique*, car les problèmes étudiés sont extrêmement concrets. Le comptage probabiliste est motivé par des besoins d'analyse qualitative des données de routeurs internet et par l'optimisation de requêtes pour des bases de données. Le hachage permet d'avoir des accès en temps constant à tous les éléments d'un grand ensemble de données, et le hachage avec pagination permet de prendre en compte de façon réaliste le fait qu'un disque magnétique soit paginé, et qu'accéder à un élément signifie accéder à une page entière d'éléments. L'algorithme de tri rapide est fondamental pour l'étude d'ensemble de données, et gagner sur sa vitesse moyenne d'exécution est une réelle motivation. Le second aspect de ce travail est l'*analyse* de ces problèmes en s'appuyant sur les méthodes de la *combinatoire analytique*. Cela signifie que les complexités annoncées sont prouvées, et ne dépendent pas d'un choix particulier d'implantation ni des jeux de données choisis pour l'expérimentation.

La contribution de l'analyse ne se limite pas à la validation de résultats pratiques déjà existants, car l'analyse est parfois au cœur même de la *conception* d'un nouvel algorithme meilleur que les précédents. Par exemple, l'écriture de l'algorithme de comptage probabiliste Loglog présenté dans le chapitre 4 repose sur la connaissance d'une certaine correction. L'analyse nous apprend que cette correction est une constante, et fournit une expression exacte ($\sqrt{2}e^{\gamma}$), que l'expérience n'aurait pu qu'approcher numériquement. Pour la version optimisée super Loglog de cet algorithme, l'intervention de l'analyse dès les premières phases de la conception de l'algorithme est encore plus spectaculaire. Tout d'abord, l'idée de cette optimisation provient de l'analyse de l'algorithme précédent, et ensuite sa mise au point dépend également d'une valeur de correction qui n'est plus une constante mais se révèle être une fonction périodique de faible amplitude et de période logarithmique !

Si l'analyse guide l'intuition pratique, il est également vrai que les simulations et les caractéristiques observées peuvent beaucoup aider la mise au point d'une preuve.

C'est cette double imbrication entre objectifs pratiques d'une part, méthodes analytiques d'autre part qui permet d'atteindre les résultats présentés ici. Ce travail peut être lu à deux niveaux.

- Pour une lecture complète, chaque chapitre présente les enjeux et les motivations du sujet, ainsi que les résultats principaux. Ensuite une large proportion du texte est consacrée à l'analyse et aux preuves de ces résultats.
- Pour un lecteur intéressé à l'algorithmique pratique, il est possible d'extraire les résultats qui sont présentés de manière à être directement utilisables, et qui sont le plus souvent possible accompagnés d'exemples et de simulations. Ainsi, le code complet des algorithmes de comptage probabilistes

Loglog et super Loglog est fourni à la fin du chapitre 5, et le théorème 4.1 et l'énoncé 5.1 donnent leur précision respective. La structure de données des listes à sauts peut être implantée sans difficulté, et les algorithmes de base sur cette structure sont rappelés ; enfin une comparaison systématique avec la structure bien connue des arbres binaires de recherche est fournie. L'algorithme d'insertion du hachage à essai aléatoire avec pagination est décrit et analysé dans le chapitre 3.

Ce qui suit présente les résultats principaux de cette thèse, selon deux directions, les résultats algorithmiques et les résultats analytiques. On organise tout d'abord les résultats algorithmiques en trois thèmes principaux[1] (Stocker-Manipuler-Extraire de l'information), puis dans une deuxième section les résultats analytiques, lesquels concernent des séries génératrices bivariées.

1 Résultats algorithmiques

1.1 Stocker

Les chapitres 2 et 3 présentent deux structures de données, qui permettent de stocker des ensembles de données. La première, la liste à sauts ou "jumplist", est une structure originellement inventée par Hervé Brönnimann et Frédéric Cazals et décrite, accompagnée de son analyse dans la publication commune [BCD03]. Cette analyse est présentée en détail dans ce manuscrit. La deuxième, est la table de hachage avec pagination, gérée par la méthode de hachage à essai aléatoire, qui est une structure classique. La contribution de cette partie du travail, réalisé en collaboration avec Alfredo Viola, est d'obtenir une mise en équation complète du comportement de l'algorithme de remplissage de la table de hachage, ce qui permet notamment d'obtenir des estimations asymptotiques précises du coût de construction de cette table.

Listes à sauts

Une liste à saut est une structure de donnée basée sur une liste chaînée triée, dans laquelle chaque nœud possède un pointeur supplémentaire, le pointeur de saut *jump* qui pointe vers un nœud ultérieur de la liste, et permet de la parcourir de manière accélérée. Ces pointeurs jumps vérifient quelques contraintes naturelles : ils ne se croisent pas (au sens strict), et forment visuellement une architecture d'arches au dessus de la liste chaînée. Les listes à sauts sont présentées dans le chapitre 2 et on peut voir un exemple de liste à saut sur la figure 2.4.

Cette structure utilise une mémoire de $2n$ pointeurs pour stocker n éléments, plus bien sûr la place mémoire nécessaire pour stocker les éléments eux-mêmes. Il est de plus possible de définir les listes à sauts randomisées[2], en s'inspirant du modèle des skiplists aléatoires. Les pointeurs jump sont alors distribués de façon uniforme, tout en respectant les contraintes de non croisement. La construction d'une liste à sauts randomisée est réalisée en temps linéaire.

Hachage à essai aléatoire

Le hachage à essai aléatoire avec pagination est un algorithme qui permet de gérer une grande base de données, en gardant un accès en temps moyen constant à chacun des éléments. Larson dans [Lar83] a montré que l'espérance du coût de construction d'une table α-pleine est linéaire en m (une table de taille m est α-pleine lorsqu'elle contient $m\alpha$ éléments). On obtient des expressions

[1] Une structure de données est bien sûr concernée par plusieurs thèmes, par exemple, une liste à sauts permet de stocker et de manipuler des éléments.

[2] L'introduction de tirages au sort dans un algorithme est appelée aléatoirisation, ou *randomisation* (selon l'usage en anglais).

asymptotiques pour l'espérance et la variance dans le cas des tables presque pleines et pour la variance dans le cas des tables α-pleines. Ces expressions sont résumées dans le théorème suivant

Théorème [Chap 3] *La série génératrice* $f(z,t,\underline{x}) = \sum_{n,k,\underline{\alpha}} f_{n,k,\underline{\alpha}} z^n t^k \underline{x}^{\underline{\alpha}}$ *qui code la répartition de la table et le coût de construction est égale à*

$$\frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} a(\underline{x},z,s)^{m-1}\left(\sum_{i=0}^{b-1}\frac{(sz/m)^i}{i!}x_i + (1-t)x_b z^b\left(e^{\frac{s}{m}} - \sum_{i=0}^{b-1}\frac{(s/m)^i}{i!}\right)\right)ds, \qquad (1)$$

où $a(\underline{x},z,s) = \sum_{i=0}^{b-1}\frac{(sz/m)^i}{i!}x_i + x_b z^b\left(e^{\frac{s}{m}} - \sum_{i=0}^{b-1}\frac{(s/m)^i}{i!}\right)$.

L'espérance du coût de construction d'une table presque pleine de taille m*, lorsque les pages sont de taille* b *est de la forme*

$$m\left(\log m + (b-1)\log\log m + \gamma - 1 + \log\frac{1}{(b-1)!} + (b-1)^2\left(\frac{\log\log m}{\log m}\right) + O\left(\frac{1}{\log m}\right)\right),$$

et la variance vérifie

$$\mathbb{V}(cc) = O(m^2).$$

La variance du coût de construction d'une table α*-pleine vérifie* $\mathbb{V} = o(m^2)$*. Par conséquent la distribution du coût de construction d'une table* α*-pleine est concentrée.*

1.2 Manipuler - Chercher

La manipulation des données informatiques repose sur l'existence de plusieurs fonctions élémentaires fondamentales. Il faut savoir insérer un élément dans un ensemble, le rechercher, et le supprimer. Dans certains cas, pour faciliter ou accélérer ces opérations, on peut vouloir manipuler un ensemble trié. Auquel cas il faut savoir trier, et/ou disposer de fonctions d'insertion/suppression qui préservent l'ordre.

L'algorithme quicksort est l'algorithme de tri en place le plus rapide, et probablement le plus utilisé. On effectue dans le chapitre 1 l'analyse de la complexité d'un quicksort optimisé proposé par Bentley et MacIlroy dans [BM93]. Les algorithmes permettant d'insérer, rechercher et supprimer dans une liste à saut sont développés en détails dans [BCD03] et leur complexité est étudiée dans le chapitre 2. Pour le hachage à essai aléatoire, seule l'insertion est étudiée en détail au chapitre 3.

Quicksort

L'algorithme de tri rapide "quicksort", inventé par Hoare en 1962 [Hoa62] fonctionne selon la méthode diviser pour régner, en divisant le tableau en deux sous-tableaux selon la position des éléments par rapport à un pivot, puis en triant ces deux sous-tableaux. Le choix d'un bon pivot conditionne en grande partie la complexité moyenne de cet algorithme, et le choix d'une stratégie différente pour les tableaux de petite taille influe sur le deuxième ordre de cette complexité.

La variante de quicksort inventée par Bentley et MacIlroy [BM93], version qui est utilisée dans certains Unix (FreeBSD), est analysée en détail dans la section 1.3. Les résultats analytiques obtenus confirment et précisent les résultats expérimentaux obtenus à l'époque. De plus cette approche analytique permet d'affiner légèrement les bornes expérimentales qui délimitent le choix des différentes stratégies selon la taille du tableau à trier.

Cet algorithme est fondé sur trois quicksorts différents, le quicksort classique, le quicksort médiane de 3 et le quicksort médiane 3-3. L'algorithme quicksort médiane de 3 sélectionne trois éléments

au hasard, et utilise comme pivot la médiane de cet échantillon. Le quicksort médiane 3-3 choisit trois échantillons de taille trois au hasard, et prend comme pivot la médiane des trois médianes de ces échantillons.

L'algorithme quicksort optimisé de Bentley et MacIlroy utilise quicksort médiane 3-3 pour les tableaux de taille > 40, médiane de 3 pour les tableaux de taille entre 7 et 39, et enfin le quicksort de base pour les tableaux de taille inférieure à 6, et cela de façon récursive. Nous obtenons

Théorème [Chap 1] *Le nombre moyen de comparaisons effectuées pour trier un tableau de taille n en utilisant le quicksort optimisé de Bentley et McIlroy est*

$$\frac{12600}{8027} n \log_e n + \left((\gamma - 1)\frac{12600}{8027} - \lambda_2\right) n + \frac{12600}{8027}\log_e n + \left(\left(\frac{1}{2} + \gamma\right)\frac{12600}{8027} - \frac{26}{3} - \lambda_2\right) + O\left(\frac{1}{n}\right),$$

où $\lambda_2 = 0.3727\ldots$ et γ est la constante d'Euler. Numériquement cette complexité vaut

$$1.5697\, n \log_e n - 1.0363\, n + 1.5697\, \log_e n - 7.3484 + O(1/n).$$

Cette complexité est à comparer avec la borne inférieure de la complexité moyenne de tout tri par comparaison qui est $\log_e n! \sim 1.4426\, n \log_e n - 1.44\, n$.

Listes à sauts

La structure de liste à sauts se manipule de manière relativement simple. Il est possible de rechercher, insérer ou supprimer un élément. La recherche d'un élément se fait en utilisant les pointeurs jump à chaque fois que c'est possible, et en marchant le long de la liste chaînée sinon. L'insertion et la suppression sont très similaires à ce qui se fait pour les listes chaînées.

La manipulation des listes à sauts aléatoires est plus délicate, et plus intéressante aussi, car il s'agit de maintenir les propriétés d'aléa qu'on impose lors de ces opérations d'insertion et de suppression. La section 2.4.3 présente l'algorithme d'insertion. La suppression, décrite dans l'article [BCD03], se fait en inversant le processus d'insertion.

Une contribution importante du chapitre 2 est de détailler le coût de chacune des manipulations qui peut être faite sur une liste à sauts aléatoire, en comparant à chaque fois ces performances à celles des arbres binaires de recherche. Le théorème suivant résume les caractéristiques des listes à sauts.

Théorème [Chap 2]

- *Il y a C_n listes à sauts de taille n, où C_n désigne les nombres de Catalan.*
- *Le coût moyen d'insertion dans une liste à sauts est $\sim 2 \log n$.*

  La profondeur d'un élément est définie comme le nombre de pointeurs qu'il faut suivre pour accéder à cet élément.
- *La longueur de cheminement interne, égale à la somme des profondeurs des nœuds est équivalente à $2n \log n$.*
- *Le profil d'une liste à sauts, défini comme la distribution de la profondeur de ses éléments, est en moyenne asymptotiquement gaussien.*
- *La hauteur d'une liste à sauts, qui est le maximum des profondeurs des nœuds, est en moyenne inférieure à la hauteur moyenne d'un arbre binaire de recherche, soit $c \log n$, où $c = 4.31\ldots$.*
- *La profondeur moyenne du k-ième élément dans une liste à sauts de taille n est équivalente à $\log k + \log(n - k)$.*

Cette structure possède quelques propriétés agréables. La recherche d'intervalle se fait aisément, de par la structure sous-jacente de liste chaînée triée. Ensuite, les éléments du début de la liste sont accessibles en un temps très court, en se contentant de suivre la liste chaînée. Cette propriété s'avère utile si les éléments sont triés selon la fréquence à laquelle ils sont recherchés.

Hachage à essai aléatoire

La recherche d'éléments dans une table de hachage se fait très facilement. D'après les résultats concernant le coût de construction, une recherche dans une table α-pleine se fait en temps moyen constant. Cette valeur se dégrade lorsque la table se remplit, jusqu'à valoir en moyenne $\log n$ lorsque la table est pleine.

1.3 Extraire de l'information

Comptage probabiliste

Comment évaluer à faible coût le nombre d'ordinateurs différents connectés à un serveur ? Quelle est la richesse du vocabulaire de Shakespeare comparée à celui de Victor Hugo ? Combien y a t il de noms de famille différents parmi tous les habitants de Paris ? et si on ajoute Boulogne ? Comment estimer le nombre d'adresses IP associées aux paquets qui circulent dans un routeur ? Comment savoir en temps réel combien de personnes ont regardé ma page web aujourd'hui ?

Toutes ces questions peuvent être résumées en une seule qui est d'estimer le nombre d'éléments distincts dans un ensemble de données dont on ignore la distribution. Ce problème fait partie d'une problématique plus générale qui vise à évaluer les répétitions dans des grands ensembles de données, et qui est présentée par Alon, Matias et Szegedy dans [AMS99]. Les résultats des chapitres 4 et 5 sont issus d'une collaboration avec Philippe Flajolet et sont publiés dans [DF03].

Les chapitres 4 et 5 décrivent les algorithmes Loglog et super Loglog qui permettent d'estimer le nombre d'éléments distincts dans un multiensemble, qu'on nomme *cardinalité*, ce en une seule passe sur les données, en utilisant une mémoire de moins d'un kilooctet de mémoire, et avec une précision moyenne d'environ 3%. Ou encore, en utilisant l'espace mémoire nécessaire à l'écriture de cette page-ci en latex, obtenir un algorithme qui estime la cardinalité avec une erreur typiquement de l'ordre de 1%.

Les applications de ces algorithmes sont suggérées par les questions posées au début de cette section. Il s'agit de pouvoir détecter une attaque sur un serveur, si elle se manifeste par un nombre de connections distinctes anormales. Ou encore de pouvoir repérer un virus ou un ver selon le nombre d'adresses IP qui propagent des paquets infectés. C'est également de pouvoir obtenir des informations très variées sur des textes, qu'ils soient issus d'œuvres littéraires ou à l'opposé, des traces de serveur HTTP.

Les algorithmes Loglog et super Loglog se basent sur les propriétés de suites binaires aléatoires, et en particulier sur la taille du plus long préfixe de zéros. Le lien entre les suites binaires aléatoires et des données réelles qui peuvent être très variées s'effectue par le hachage. Il est en effet très réaliste de dire qu'une bonne fonction de hachage permet d'obtenir une distribution presque indistinguable de l'uniformité. Le traitement effectué par ces algorithmes est très rapide, et n'utilise que peu de mémoire (les éléments ne sont jamais stockés !). Il est donc possible de les utiliser pour des applications où le flux de données est très important. Estan, Fisk et Varghese [EVF03] rapportent le cas de données récoltées au niveau d'un routeur pour repérer un ver, ce au rythme de 0.5GB de données comprimées par heure.

Ces algorithmes permettent d'obtenir une estimation à chaque instant du cardinal des éléments déjà lus. De plus ils peuvent être implantés sous forme distribuée.

Le théorème suivant repose sur un modèle abstrait qui utilise une fonction de hachage statistiquement "parfaite". Une telle fonction peut être extrêmement bien imitée dans la réalité, ce qui justifie le modèle.

Théorème [Chap 4] *Soit E_n la valeur retournée par l'algorithme Loglog appliqué à un multiensemble de cardinal inconnu n. Alors*

(i) L'estimée E_n est asymptotiquement non biaisée, c'est-à-dire que lorsque $n \to \infty$,

$$\frac{1}{n}\mathbb{E}(E_n) = 1 + \theta_{1,n} + o(1), \qquad \text{où} \quad |\theta_{1,n}| < 10^{-6}.$$

(ii) L'erreur standard, définie comme $\frac{1}{n}\sqrt{\mathbb{V}(E_n)}$ vérifie quand $n \to \infty$,

$$\frac{1}{n}\sqrt{\mathbb{V}(E_n)} = \frac{\beta_m}{\sqrt{m}} + \theta_{2,n} + o(1), \qquad \text{où} \quad |\theta_{2,n}| < 10^{-6}.$$

où les valeurs de β_m sont proches de 1.3, et sont décrites précisément dans le théorème 4.1 du chapitre 4.

L'algorithme super Loglog est basé sur les mêmes principes algorithmiques que l'algorithme Loglog, mais avec une optimisation supplémentaire. Il est également non-biaisé, et a une erreur relative majorée par $\frac{0.95}{\sqrt{m}}$ ce qui est un progrès d'environ 30% par rapport à la version Loglog. Il est de plus valable sur un intervalle de valeurs extrêmement étendu.

2 Résultats analytiques

Les résultats analytiques obtenus sont des développements asymptotiques de coefficients de séries génératrices bivariées à singularités confluentes. Ces séries génératrices interviennent dans divers problèmes issus de la combinatoire analytique ou de l'analyse d'algorithmes.

Généralement la variable z "code" la taille n de l'ensemble étudié. Lorsqu'on veut ajouter un ou plusieurs paramètres, on va de la même manière associer à chacun une variable, et l'on obtient alors des séries génératrices multivariées.

Lors de l'analyse des listes à sauts, j'ai été conduite à étudier une série génératrice bivariée qui contient des termes de la forme $(1-z)^{-a}(1-zu)^{-b}$. Une partie du chapitre 2 est consacrée à l'estimation de l'asymptotique des coefficients de telles séries génératrices. Une extension naturelle de cette problématique est l'estimation de l'asymptotique des coefficients de séries génératrices de la forme $(1-z)^{-a}(1-zu)^{-b}(1-u)^{-c}$. La section 2.6 présente les résultats asymptotiques recherchés.

Des séries de ce type interviennent dans des problèmes variés issus de l'analyse d'algorithmes, dont trois sont présentés en détail. Tout d'abord les listes à sauts, qui sont pour moi l'origine de cette problématique, ensuite les algorithmes de sélection rapide (quickselect) et quicksort partiel, et enfin une étude de fréquence d'apparition de symboles sous un modèle de source réductible, dite non-primitive.

Les résultats analytiques obtenus dans le chapitre 2 sont résumés dans le théorème suivant. Soit la fonction génératrice bivariée f définie par

$$f(z,u) = \left(\frac{1}{1-z}\right)^a \left(\frac{1}{1-zu}\right)^b \left(\frac{1}{1-u}\right)^c,$$

et soit $w(z,u)$ une fonction analytique dans un domaine suffisant (par exemple pour $|u| \leq 2$ et $z \leq 2$), et telle que $w(1,1) = 1$. On montre le résultat très général suivant

Théorème [Chap 2] *Si les trois variables a, b et c vérifient $\Re(a) > 0$, $\Re(b) > 0$ et $\Re(c) > 0$, et que n, k et $n-k$ sont proportionnels, avec $k < n$, alors*

$$[z^n u^k] f(z,u) w(z,u) \sim \frac{k^{b+c-1} n^{a-1}}{\Gamma(a)} \frac{1}{\Gamma(b+c)} {}_2F_1\left(1-a, b; b+c; \frac{k}{n}\right),$$

où ${}_2F_1$ est une fonction hypergéométrique.

On obtient le même type de résultats si on ajoute à la série génératrice des termes logarithmiques, ou de faible variation.

3 Organisation

Le chapitre 1 est consacré à des rappels sur les méthodes analytiques utilisées dans cette thèse, en particulier la transformation de Mellin. Une partie de ces techniques est mise en œuvre pour l'analyse d'un algorithme de quicksort optimisé qui est présenté dans la section 1.3. Cette analyse est publiée dans l'article

– "Asymptotic Analysis of an Optimized Quicksort Algorithm" publié dans le journal *Information Processing Letters* en 2003, volume 85, pages 73–77 [Dur03].

Le travail réalisé en collaboration avec Stephen Taylor sur le comportement des arbres binaires de recherche dans un modèle d'insertion/suppression symétrique est également une mise en œuvre des méthodes analytiques présentées dans le chapitre de rappels. Ce travail n'est pas détaillé dans cette thèse. Il a donné lieu a la publication

– "Emerging Behavior as Binary Search Trees Are Symetrically Updated" dans le journal *Theoretical Computer Science* en 2003, volume 297, pages 425–445 [DT03].

Le chapitre 2 étudie d'une part la structure de données des listes à sauts, et d'autre part un problème général d'asymptotique bivariée qui s'applique à l'asymptotique de paramètres de listes à sauts. Le travail sur les listes à sauts a été réalisé en collaboration avec Hervé Brönnimann et Frédéric Cazals et est l'objet d'un article.

– "Randomized Jumplists : A Jump-and-Walk Dictionary DataStructure" publié dans le compte-rendu de la conférence STACS'03, pages 283–294 [BCD03].

Le chapitre 3 s'intéresse à l'algorithme de hachage à essai aléatoire. On présente tout d'abord l'algorithme, ainsi que ses motivations, puis on s'intéresse particulièrement au paramètre qu'est le coût de construction de la table. Il y a alors deux cas de figure étudiés. Dans le premier, la table est pleine, et on obtient la moyenne et la variance du coût de construction. Dans le deuxième la table n'est pas pleine, ce qui est le cas souhaité en pratique, et on obtient l'espérance du coût de construction. Ce travail est réalisé en collaboration avec Alfredo Viola et fera l'objet d'un article ultérieur.

Les deux derniers chapitres sont consacrés à deux algorithmes de comptage probabiliste, l'algorithme Loglog et sa variante optimisée super Loglog. Ce travail est réalisé en collaboration avec Philippe Flajolet, et a donné lieu à la publication

– "Loglog Counting of Large Cardinalities", publié dans le compte-rendu de la conférence "European Symposium on Algorithms" (ESA03) en 2003, pages 605–617 [DF03].

Le chapitre 4 présente un survol des différentes méthodes de comptage probabilistes, et analyse l'algorithme d'estimation de cardinal nommé comptage par collisions (linear counting). La section 4.4 est consacrée à l'énoncé de l'algorithme Loglog, et la section 4.5 à son analyse.

Le chapitre 5 est la suite du chapitre précédent, et présente l'algorithme super Loglog et son analyse. Il y a ensuite trois sections consacrées à la mise au point de l'algorithme vis à vis de données réelles (correction des non-linéarités initiales) et au calcul de la précision obtenue dans ce cas. La section 5.6 contient les résultats expérimentaux obtenus sur des données diverses et variées, la section suivante 5.7 montre comment améliorer le coût mémoire de l'algorithme en compressant les données stockées pendant l'exécution de l'algorithme. Enfin la dernière section fournit le code source des algorithmes Loglog et super Loglog, écrit en C.

Chapitre 1

Quelques méthodes de la combinatoire analytique

Sommaire

Cette thèse étudie différents aspects de l'algorithmique des ensembles de données, en utilisant des outils issus de la combinatoire analytique. Un certain nombre de méthodes sont à la base des différentes analyses menées dans cette thèse, et ce chapitre de rappels en fait un tour d'horizon.

La première section présente les principaux outils d'analyse univariée, ainsi que leur champ d'application. On voit tout d'abord la notion d'holonomie (voir aussi [Dur00]), qui permet dans un grand nombre de cas de formaliser le passage d'un problème combinatoire à un problème d'analyse de séries génératrices. Cette mise en équation peut être faite par d'autres moyens, par exemple de manière directe comme c'est le cas dans le chapitre 4.

Ensuite, les méthodes analytiques qui permettent d'étudier ces séries génératrices sont l'analyse de singularités, la méthode de col, la poissonisation, et enfin la transformation de Mellin. La transformation de Mellin étant particulièrement utile pour l'analyse de l'algorithme Loglog, la deuxième section lui est entièrement consacrée. On rappelle notamment les relations entre asymptotique de la fonction et singularités de sa transformée.

La troisième section sert d'étude de cas consacrée à l'analyse d'un quicksort optimisé mis au point par Bentley et McIlroy. Cette analyse originale [Dur03] est l'occasion de montrer dans son intégralité le processus d'analyse d'algorithme, du problème combinatoire à l'asymptotique du paramètre étudié.

1.1 L'analyse d'algorithmes univariée

L'ensemble de données que l'algorithme va permettre de stocker ou de manipuler est généralement imprévisible, cependant on souhaite avoir des informations sur le coût (au sens large) d'une exécution. Typiquement, on souhaite pouvoir répondre aux questions : combien de temps faut-il pour trier un tableau de taille n, combien de pointeurs faut-il suivre pour retrouver un élément dans une structure de données (arbre binaire de recherche, liste à sauts) ? Combien de pages charge-t-on

pour atteindre l'élément recherché dans une table de hachage avec pagination ? La réponse à ces questions varie bien sûr selon l'ensemble de données étudié ou l'élément recherché.

Pour répondre au mieux à ces questions, la stratégie est d'étudier tous les ensembles de données possibles de taille n, considérés comme équiprobables. Le paramètre auquel on s'intéresse est alors vu comme une variable aléatoire X, dont la loi de probabilité est la distribution des valeurs du paramètre étudié sur tous les ensembles de données. La réponse à la question "Combien vaut le paramètre sur tel ensemble de taille n ?" est alors : "Ce paramètre suit telle loi de probabilité sur les ensembles de taille n".

L'étude de cette distribution passe d'abord par l'analyse des quantités caractéristiques clés que sont la moyenne et la variance. La moyenne du paramètre sur des ensembles de taille n donne lieu à une séquence μ_n (et la variance à une séquence σ_n^2), qui peut être codée par une série génératrice univariée $\mu(z) := \sum \mu_n z^n$. De façon inverse, la valeur μ_n s'obtient en fonction de la série génératrice $\mu(z)$ par une intégrale de Cauchy $[z^n]\mu(z) = \mu_n = \frac{1}{2i\pi} \oint \mu(z) z^{-n-1}\, dz$.

Dans cette thèse, le schéma d'étude utilisé comprend deux étapes, la première est de traduire le problème sous forme d'une équation vérifiée par la série génératrice du paramètre ($\mu(z)$ dans l'exemple de la moyenne). Cette traduction se base généralement sur des propriétés de décomposabilité de la structure de données sous-jacente, laquelle se traduit en une récurrence sur la séquence et enfin en équation fonctionnelle sur la série génératrice. Dans le cas holonome, cette équation fonctionnelle se révèle être une équation différentielle linéaire à coefficients polynomiaux. Cette partie est développée dans la sous-section "Holonomie". La seconde étape de ce schéma d'étude consiste à extraire de la série génératrice des informations sur l'asymptotique des coefficients. Cette étape se base (dans cette thèse) sur trois méthodes exposées plus loin, qui sont l'analyse de singularité, la transformation de Mellin et la méthode de col.

Ce schéma d'étude conduit à des résultats asymptotiques sur les paramètres étudiés pour toutes les applications présentées dans cette thèse.

1.1.1 Holonomie

La combinatoire du problème se traduit fréquemment en une récurrence sur le paramètre qui détermine la complexité. Cela provient du fait que les structures étudiées sont souvent décomposables : arbres binaires, tableaux, listes à sauts. Cette décomposition se transmet souvent aux paramètres, taille d'un arbre binaire, longueur de cheminement interne d'une liste à sauts, et se transmet à la récurrence satisfaite par le paramètre. Cette récurrence est dans de nombreux cas linéaire à coefficients polynomiaux, et on dit alors qu'elle est holonome [Dur00], ou encore [Chy98, Sta80].

La série génératrice de la séquence f_n est définie selon les cas par $f(z) = \sum f_n z^n$ qui se nomme série génératrice ordinaire ou par $f(z) = \sum f_n \frac{z^n}{n!}$, nommée série génératrice exponentielle. La récurrence obtenue pour la séquence f_n est traduite en équation différentielle linéaire à coefficients polynomiaux satisfaite par $f(z)$. Les conditions initiales de l'équation différentielle sont obtenues simplement à partir des conditions initiales de la récurrence. Cette traduction peut être réalisée automatiquement dans certains cas [SZ94], et ne présente pas de difficulté majeure pour les exemples présentés dans cette thèse.

Cette équation différentielle est de degré un dans de nombreux cas. La méthode de variation de la constante fournit alors une expression intégrale de la série génératrice. La solution obtenue n'est pas toujours particulièrement avenante, mais elle possède souvent un développement simple au voisinage de sa singularité dominante. Cette singularité est en pratique très souvent égale à 1, pour tous les paramètres ayant une croissance polynomiale. Cette approche permet donc d'obtenir la série génératrice recherchée, et souvent un équivalent de cette série au voisinage de sa singularité

principale.

Une autre méthode est d'obtenir la traduction du problème combinatoire en équation fonctionnelle sur une série génératrice de façon directe, de par la connaissance des distributions de probabilité sous-jacentes. Cette approche directe passe par la connaissance de relations entre des opérations combinatoires et des transformations sur les séries génératrices associées. C'est la méthode utilisée pour l'analyse des algorithmes Loglog et super Loglog, ainsi que pour l'analyse du hachage à essai aléatoire. Par exemple, pour la mise en équation de la moyenne de l'algorithme Loglog, on exprime tout d'abord la série génératrice de probabilité relative à la valeur d'un registre. Puis cette série est élevée à la puissance m pour exprimer la prise en compte des m registres.

Par ces deux méthodes, on obtient une expression de la série génératrice recherchée, éventuellement sous forme intégrale. Pour accéder à l'information qui nous intéresse sur les structures de taille n, il faut alors extraire l'asymptotique du coefficient de $[z^n]$ dans la série génératrice dont on a obtenu une expression.

1.1.2 Méthodes analytiques

Cette section résume les principales méthodes analytiques utilisées pour extraire de l'information asymptotique relative aux coefficients d'une série génératrice. Ces méthodes sont a priori uniquement valables pour des séries génératrices univariées.

L'analyse de singularités

L'analyse de singularité, due à Flajolet et Odlyzko [FO90], permet d'obtenir l'asymptotique des coefficients d'une série génératrice, lorsqu'on dispose d'un équivalent adapté de cette série au voisinage de sa singularité dominante.

Définition 1.1 *On dit qu'un ouvert D du plan complexe est un* Δ-domaine *s'il existe deux réels $R > 1$, et $0 < \phi < \pi/2$ tels que*

$$D = \{z \,||z| < R, z \neq 1, |Arg(z-1)| > \phi\}.$$

Une fonction est Δ-analytique *si elle est analytique sur un Δ-domaine.*

Méthode d'analyse de singularités. *Soit f une fonction Δ-analytique, telle que dans l'intersection de son Δ-domaine et d'un voisinage de 1 on ait*

$$f(z) \sim \frac{1}{(1-z)^a} \log^b \left(\frac{1}{1-z}\right),$$

alors

$$[z^n] f(z) \sim \frac{n^{a-1}}{\Gamma(a)} \log^b n.$$

En pratique, l'équivalent de la série génératrice étudiée, f, est souvent composé de fonctions de type $(1-z)^\alpha \log^\beta(1-z)$, avec α et β des petits entiers, ce qui fait que les résultats de la figure 1.1 permettent de faire ce passage de série génératrice à asymptotique des coefficients, jusqu'à un $o(1)$, dans de nombreux cas.

Cette méthode est rappelée dans la section 2.2.1, et est utilisée de nombreuses fois et étendue dans le chapitre 2.

$f(z)$	f_n
1	$0 + o(1)$
$\frac{1}{1-z}$	$1 + o(1)$
$\frac{1}{(1-z)^2}$	$n + 1 + o(1)$
$\frac{1}{1-z}\log\frac{1}{1-z}$	$\log n + \gamma + o(1)$
$\frac{1}{(1-z)^2}\log\frac{1}{1-z}$	$n\log n + (\gamma-1)n + \log n + 1/2 + \gamma + o(1)$
$\frac{1}{(1-z)^2}\log^2\frac{1}{1-z}$	$n\log^2 n + 2(\gamma-1)n\log n + \gamma^2 n - 2\gamma n + 2n - \frac{\pi^2 n}{6} + o(n)$

FIG. 1.1 – Récapitulatif de l'asymptotique des coefficients de certaines fonctions de base.

Méthode de col

Un point col pour une fonction complexe est l'équivalent d'un col en montagne. De même que le randonneur choisit de passer par un col pour changer de vallée en minimisant son effort, un contour d'intégration peut choisir de passer par un point col, pour concentrer à ce point la valeur de l'intégrale.

La méthode de col consiste alors à écrire le coefficient de z^n de la série génératrice étudiée grâce à une intégrale de Cauchy, puis à choisir comme contour d'intégration un cercle qui passe par le point col de l'intégrande. Ce point col se détermine explicitement car c'est un zéro de la dérivée de l'intégrande. Sous certaines hypothèses, la contribution dominante est alors concentrée autour du point col, et peut alors être approximée. La méthode de col est rappelée de manière très détaillée par Szpankowski dans [Szp01, p344], et est survolée ici.

Méthode du col. *Soit une fonction $g(z)$, à coefficients de Taylor positifs, de rayon de convergence ρ, et soit $h(z) = \log g(z) - n\log z$. Soit r l'unique racine (le point col) de l'équation $h'(r) = 0$, alors sous des hypothèses de croissance adaptées pour h, on a*

$$[z^n]g(z) \sim \frac{g(r)r^{-n}}{\sqrt{2\pi h''(r)}}.$$

La méthode de Laplace est une version réelle de la méthode de col, qui relie l'asymptotique d'une intégrale à la position du maximum de l'intégrande, sous certaines hypothèses.

La méthode de col est à la base des théorèmes de dépoissonisation analytique, qu'on présente maintenant.

Poissonisation

Lorsque les méthodes précédentes ne s'appliquent pas, comme c'est le cas dans l'analyse de l'algorithme Loglog, il peut s'avérer très difficile d'extraire l'asymptotique de coefficients d'une série génératrice.

Le problème peut alors être étudié sous un modèle de Poisson (d'où le nom de poissonisation). Il s'agit de s'intéresser à l'asymptotique du paramètre étudié sur un ensemble dont la taille suit une loi de Poisson au lieu d'être fixée. Intuitivement, le fait de relâcher la contrainte de la taille fixe permet de gagner en liberté, et d'utiliser des méthodes qui n'étaient pas disponible auparavant. On définit la série génératrice de Poisson d'une séquence f_n comme étant $\tilde{f}(z) = \sum f_n \frac{z^n}{n!} e^{-z}$, qui est la série génératrice exponentielle, à un facteur e^{-z} près.

Lorsqu'on a obtenu des résultats asymptotiques dans ce modèle Poisson, on peut, si les résultats ont de bonnes propriétés de croissance, obtenir des résultats asymptotiques pour le modèle où la

taille est fixe. C'est la dépoissonisation.

Le principal résultat de la poissonisation/dépoissonisation utilisé dans ce manuscrit est qu'elle transforme un problème d'extraction de coefficient en un problème d'évaluation de fonction. Plus précisément, si $\tilde{f}(z)$ est la série génératrice de Poisson, sous certaines hypothèses, on a $f_n \sim \tilde{f}(n)$. C'est la dépoissonisation analytique, due à Philippe Jacquet et Wojciech Szpankowski [Szp01].

Ce processus de poissonisation/dépoissonisation est utilisé pour l'analyse de l'algorithme Loglog. Des références pour cette technique sont [JS98, Szp01].

Transformation de Mellin

La transformation de Mellin est un outil qui permet de relier l'asymptotique d'une fonction en 0 ou l'infini à l'ensemble des singularités de sa transformée dans le plan complexe. De plus la transformation de Mellin se manipule très naturellement pour toute une classe de sommes (les sommes harmoniques), qui interviennent fréquemment en analyse d'algorithmes. Ces propriétés en font une méthode privilégiée en analyse d'algorithmes, et un complément naturel de la poissonisation. En effet, la poissonisation exprime un problème combinatoire, ou d'analyse d'algorithme sous forme d'une évaluation de fonction en n, avec n qui tend vers l'infini, et la transformation de Mellin permet de relier cette évaluation à une connaissance des singularités d'une transformée, qui se calcule parfois facilement.

La section suivante est consacrée à la présentation et à l'étude de la transformation de Mellin, qui est utilisée à plusieurs reprises lors de l'analyse de l'algorithme Loglog.

1.2 Transformation de Mellin

La transformation de Mellin, introduite par Hjalmar Mellin(1854-1933), est une transformation de fonctions qui permet de relier le développement asymptotique d'une fonction à l'ensemble des singularités de sa transformée

Définition 1.2 *La transformation de Mellin d'une fonction f définie sur l'axe réel est la fonction complexe $f^*(s)$, où*

$$f^*(s) = \int_0^\infty f(t) t^{s-1}\, dt.$$

1.2.1 Propriétés principales

Bande fondamentale

La transformée de Mellin de la fonction $f(t)$ est définie par une intégrale. Cette intégrale a un sens pour $s > \alpha$, si au voisinage de 0, $f(t) = O(t^{-\alpha})$, et pour $s < \beta$ si au voisinage de l'infini $f(t) = O(t^{-\beta})$. Ceci conditionne l'existence de l'intégrale à l'ordre de grandeur de f aux bornes de l'intégrale. La proposition suivante formalise le domaine de définition d'une transformée de Mellin.

Proposition 1.1 *Il existe un intervalle ouvert $\Omega = \{z | \Re(z) \in]\alpha, \beta[\}$ maximal tel que la transformée de Mellin de f est définie si $\alpha < \Re(s) < \beta$. Cette bande de définition est appelée* bande fondamentale *et est notée $\langle \alpha, \beta \rangle$.*

La valeur α est définie comme l'inf de tous les a tels que $f(x) = O(x^{-a})$ en 0. De même, $\beta = \sup\{b | f(x) = O(x^{-b}), x \to \infty\}$. La transformée de Mellin a une zone de définition si $\alpha < \beta$, sinon l'intégrale est nulle part convergente, et on dit que la fonction n'a pas de transformée de Mellin. Par exemple, les constantes et les polynômes n'ont pas de transformée de Mellin.

Exemples

L'exemple le plus immédiat est la fonction exponentielle e^{-x} qui a, par définition, comme transformée de Mellin la fonction Γ. Plus généralement, il y a certaines fonctions qui apparaissent fréquemment, et dont on donne ici les transformées de Mellin.

Fonction	Transformée	Bande fond.
e^{-x}	$\Gamma(s)$	$\langle 0, \infty\rangle$
$e^{-x}-1$	$\Gamma(s)$	$\langle -1, 0\rangle$
$\frac{1}{1+x}$	$\frac{\pi}{\sin \pi s}$	$\langle 0, 1\rangle$
$\log(1+x)$	$\frac{\pi}{s \sin \pi s}$	$\langle -1, 0\rangle$

Il y a beaucoup d'autres exemples classiques qui ne sont pas cités ici, et qui peuvent être trouvés dans [FGD95].

Propriétés fonctionnelles - Séries harmoniques

La transformation de Mellin satisfait un certain nombre de propriétés fonctionnelles qui permettent, jointes aux exemples précédents, de calculer très facilement la transformée de Mellin d'un grand nombre de fonctions.

Certaines de ces propriétés sont listées dans le tableau suivant, elles se démontrent toutes assez facilement par des manipulations simples d'intégrales.

Fonction	Transformée	Bande fond.	Conditions
$f(x)$	$f^*(s)$	$\langle \alpha, \beta\rangle$	
$f(\mu x)$	$\mu^{-s} f^*(s)$	$\langle \alpha, \beta\rangle$	$\mu > 0$
$f(x^\rho)$	$\frac{1}{\rho} f^*(\frac{s}{\rho})$	$\langle \rho\alpha, \rho\beta\rangle$	$\rho > 0$
$\frac{d}{dx} f(x)$	$(1-s) f^*(s-1)$	$\langle \alpha', \beta'\rangle$	
$f(x) \log x$	$\frac{d}{ds} f^*(s)$	$\langle \alpha, \beta\rangle$	

Exemple. Ces propriétés permettent de déduire la transformée de Mellin de la fonction $e^{-x/m}$, qui est très présente dans l'analyse de l'algorithme de comptage probabiliste Loglog(voir section 4.5.4). La transformée de Mellin de $e^{-x/m}$ est égale à $m^s \Gamma(s)$.

On peut déduire de ces propriétés fonctionnelles, et de la linéarité de l'intégrale, la transformée de Mellin d'une somme harmonique. On obtient alors

$$\left(\sum \lambda_k f(\mu_k x)\right)^* = \left(\sum \lambda_k (\mu_k)^{-s}\right) f^*(s).$$

La bande fondamentale de cette transformée est l'intersection de la bande fondamentale de f, avec la bande d'existence de $\sum \lambda_k (\mu_k)^{-s}$. On a ainsi pu "factoriser" f^* dans la transformée de Mellin, ce qui permet ensuite de séparer un certain nombre de problèmes, dont la recherche de singularités.

Transformation de Mellin inverse

Il existe une transformation inverse de la transformation de Mellin, qui est décrite dans cette proposition.

Proposition 1.2 *Si $f(x)$ est une fonction continue, localement intégrable, dont la transformée de Mellin est définie dans la bande fondamentale $\langle\alpha,\beta\rangle$, alors, pour tout c appartenant à l'intervalle $]\alpha,\beta[$, on a*

$$f(x) = \frac{1}{2i\pi}\int_{c-i\infty}^{c+i\infty} f^*(s)x^{-s}\,ds$$

La preuve de cette proposition peut être trouvée dans [Wid41].

Cette transformation inverse n'est pas utilisée fréquemment, et il est en général beaucoup plus commode de reconnaître une transformée de Mellin grâce aux propriétés fonctionnelles et aux exemples déjà connus que de la calculer.

La proposition 1.2 est valable pour x réel. Pour obtenir un développement asymptotique pour $f(z)$, avec z complexe, il peut être intéressant de définir $g(x) = f(xe^{i\theta})$. On peut alors appliquer les résultats donnés par la transformation de Mellin à la fonction g, qui est définie sur l'axe réel. Cette méthode permet en pratique d'obtenir des extensions analytiques au voisinage de l'axe réel, et est utilisée dans l'analyse de l'algorithme Loglog, dans la section 4.5.5.

Propriétés asymptotiques

La transformation de Mellin est largement utilisée dans ce manuscrit car elle exprime le développement asymptotique de la fonction f en 0 ou l'infini (qu'on recherche), en fonction de singularités de f^* (qu'on connaît souvent). On introduit tout d'abord la notation suivante $\phi(s) \asymp \sum \Delta_{s_0}(s)$ signifie que chaque $\Delta_{s_0}(s)$ est une troncature de la série de Laurent de ϕ en s_0.

La correspondance entre asymptotique de f et singularités de f^* est décrite par les deux théorèmes qui suivent.

Théorème 1.1 *Soit la fonction $f(x)$ et sa transformée de Mellin $f^*(s)$ avec comme bande fondamentale $\langle\alpha,\beta\rangle$, qu'on suppose non vide.*
Si f(x) a le développement asymptotique en l'infini

$$f(x) = \sum_{\xi,k} c_{\xi,k}x^{\xi}\log^k(x) + O(x^{\delta}),$$

où la somme est finie, et où ξ vérifie $\beta \le -\xi < -\delta$. Alors $f^*(s)$ est prolongeable en une fonction méromorphe dans la bande $\langle\alpha,-\delta\rangle$, et

$$f^*(s) \asymp -\sum_{\xi,k} c_{\xi,k}\frac{(-1)^k k!}{(s+\xi)^{k+1}}$$

La preuve de ce théorème peut être trouvée dans [FGD95]. Elle est essentiellement basée sur le calcul de la transformée de Mellin de chacun des termes, qui donne la correspondance asymptotique-singularité :

$f(x)$	$f^*(s)$
$x^{\xi}\log^k(x) \quad (x \to \infty)$	$-\frac{(-1)^k k!}{(s+\xi)^{k+1}}$

Le second théorème est la réciproque du premier : il permet de passer des singularités de f^* à l'asymptotique de f. Ce résultat est utilisé à plusieurs reprises dans cette thèse.

Théorème 1.2 *Soit la fonction $f(x)$ continue sur $]0,+\infty[$ et sa transformée de Mellin $f^*(s)$ ayant comme bande fondamentale $\langle\alpha,\beta\rangle$, qu'on suppose non vide.*

Si $f^*(s)$ possède un prolongement méromorphe dans la bande $\langle \alpha, \delta \rangle$, avec $\delta > \beta$ et analytique sur $\Re(s) = \delta$, si de plus pour un $\eta \in (\alpha, \beta)$ et un $r > 1$, on a $f^*(s) = O(|s|^{-r})$ quand $s \to \infty$ avec $\eta \leq \Re(s) \leq \delta$, et si

$$f^*(s) \asymp -\sum_{\xi,k} d_{\xi,k} \frac{1}{(s-\xi)^k}$$

lorsque $s \in \langle \eta, \delta \rangle$. Alors le développement de f à l'infini peut s'écrire

$$f(x) = -\sum_{\xi,k} d_{\xi,k} \frac{(-1)^{k-1}}{(k-1)!} x^{-\xi} \log^{k-1}(x) + O(x^{-\delta}).$$

Ce théorème est appliqué à un exemple dans la section suivante.

1.2.2 Maximum de variables géométriques

On considère la variable aléatoire géométrique qui est la position du premier symbole "1" dans un mot binaire aléatoire uniforme non biaisé. (Dans un tel mot, chaque caractère vaut 0 ou 1 avec probabilité 1/2, indépendamment des autres caractères.) La probabilité que le premier 1 soit situé strictement au delà du k-ième caractère est $\frac{1}{2^k}$, car l'événement signifie que les k premiers caractères sont des 0. La probabilité que cette position soit $\leq k$ est par conséquent $(1 - 2^{-k})$. On s'intéresse au maximum des positions des premiers 1 dans un ensemble de n mots, tous binaires aléatoires non biaisés. Le maximum, noté M, est inférieur à k si toutes les positions du premier 1 sont inférieures à k, ce qui a une probabilité $(1 - 2^{-k})^n$. La loi de probabilité de M est donc

$$\Pr(M > k) = 1 - (1 - 2^{-k})^n.$$

L'espérance de cette variable aléatoire M est alors égale à

$$\mathbb{E}(M) = 1 + \sum_{k \geq 1} 1 - (1 - 2^{-k})^n.$$

Pour calculer cette espérance, on introduit la fonction $F(x)$ définie par

$$F(x) := \sum_{k \geq 1} 1 - (1 - 2^{-k})^x = \sum_{k=1}^{\infty} \left(1 - e^{-\mu_k x}\right).$$

avec $\mu_k = -\log(1 - 2^{-k})$. La fonction F est une somme harmonique, comme on le voit dans le terme droit de l'équation précédente. Sa transformée de Mellin peut alors être calculée sans difficulté, et on obtient

$$F^*(s) = -\Gamma(s)\Lambda(s), \qquad \Lambda(s) = \sum_{k=1}^{\infty} \mu_k^{-s}.$$

La bande fondamentale de $F^*(s)$ est $\langle -1, 0 \rangle$. En effet, l'approximation $(1-2^{-k})^x \sim 1 + x \log(1-2^{-k})$ au voisinage de 0 montre que $F(x) =_{x \to 0} O(x)$. Au voisinage de l'infini, on montre que $F(x) =_{x \to \infty} O(\log^2 x)$ en découpant la somme en deux parties, à savoir, la partie $k < \log^2 x$, où les termes sont majorés grossièrement par 1, et le reste, qui tend vers 0 lorsque x tend vers l'infini. Cela montre que $F(x) =_{x=\infty} O(x^\epsilon)$, quel que soit $\epsilon > 0$, et termine la caractérisation de la bande fondamentale.

Pour obtenir l'asymptotique de $F(x)$ quand x tend vers l'infini, on étudie les singularités de sa transformée de Mellin à droite de la bande fondamentale, c'est-à-dire à droite de 0. Au voisinage de $x = 0$, on a le développement

$$(-\log(1-x))^{-s} = \frac{1}{x^s}\left(1 - \frac{s}{2}x + \frac{1}{24}s(3s-5)x^2 + O(x^3)\right).$$

Ce résultat est alors appliqué à chacun des μ_k, ce qui permet d'écrire $\Lambda(s)$ sous la forme

$$\Lambda(s) = \frac{2^s}{1-2^s} - \frac{s}{2}\frac{2^{s-1}}{1-2^{s-1}} + R(s),$$

où $R(s)$ est une fonction analytique sur $\langle -\infty, 2\rangle$. Les singularités de $\Lambda(s)$ sont alors situées aux points $\frac{2ik\pi}{\log 2}$, $1 + \frac{2ik\pi}{\log 2}$, ou bien sont de partie réelle supérieure ou égale à 2. La seule singularités de la fonction $\Gamma(s)$ à droite de la bande fondamentale est le point 0.

La correspondance entre les singularités de la transformation de Mellin et l'asymptotique de la fonction permet d'énoncer la proposition :

Proposition 1.3 *La fonction $F(x) = \sum_{k\geq 1} 1 - (1-2^{-k})^x$ possède le développement asymptotique suivant lorsque $x \to \infty$*

$$F(x) = \log_2 x + P(\log_2 x) + o(1),$$

où P est une fonction périodique de période 1, qui s'écrit

$$P(x) = \frac{\gamma}{\log 2} - \frac{1}{2} + \frac{1}{\log 2}\sum_{k\in\mathbb{Z}^*} \Gamma\left(\frac{2ik\pi}{\log 2}\right) e^{-2ik\pi x}.$$

1.3 Quicksort optimisé

En 1993, Jon Bentley et Douglas McIlroy ont implanté un nouvel algorithme de tri construit à partir de plusieurs quicksorts. Leur motivation était que les algorithmes de tri existants étaient parfois très lents sur des tableaux particuliers, et aussi le fait que la complexité moyenne pouvait être améliorée. Leur algorithme mélange trois quicksorts différents, le quicksort de base, le quicksort médiane de trois et le quicksort médiane 3-3 qui sont expliqués plus tard. Ces trois stratégies sont utilisées sur des tableaux de taille différentes. L'idée directrice est que plus le tableau à trier est grand, plus on recherche un pivot centré, quitte à ce que ce choix soit plus coûteux.

Cette section étudie la complexité moyenne de l'algorithme définie comme le nombre moyen de comparaisons nécessaires pour trier un tableau rempli avec n valeurs distinctes, toutes les permutations étant prises équiprobables. L'analyse sous le modèle des permutations aléatoires possède le mérite de fournir une échelle souple et standard de comparaison des algorithmes de tri. On obtient facilement que le terme du premier ordre de cette complexité est $O(n \log n)$, mais on veut aussi obtenir le terme du deuxième ordre qui ne doit pas être négligé car cet algorithme de tri doit aussi être efficace sur des tableaux de taille moyenne. Bentley et McIlroy ont effectué des mesures expérimentales de cette complexité, qui sont en accord avec les résultats théoriques obtenus ici.

Cet algorithme de type quicksort, qui est actuellement l'algorithme de tri de FreeBSD, possède des optimisations destinées à réduire le nombre d'affectations. Les tableaux remplis avec d'éventuelles répétitions sont également pris en compte dans l'implémentation. Pour le quicksort de base, l'analyse avec des clés répétées est faite par Sedgewick [Sed77].

L'algorithme

L'algorithme de tri quicksort a été inventé en 1960 par C. Hoare [Hoa62] (anobli depuis, quicksort mène vraiment à tout). Différentes versions de cet algorithme sont présentées dans [BM93, Knu98, Sed88]. L'algorithme quicksort trie un tableau de la façon suivante : un pivot est choisi dans le tableau, puis l'algorithme compare tous les éléments (sauf le pivot) au pivot. Ceci divise le tableau en deux, le tableau des élément inférieurs au pivot, et celui des éléments supérieurs au pivot. Ces deux sous tableaux sont ensuite triés récursivement.

Le choix du pivot est essentiel, car plus il est central (c'est-à-dire proche de la valeur médiane des éléments), plus les deux sous tableaux ont des tailles voisines, et donc plus le découpage est efficace. Cependant, pour que la complexité reste bonne, ce choix doit être rapide. L'idée la plus simple est le quicksort de base, où un élément aléatoire est choisi comme pivot (par exemple le premier du tableau). Une deuxième méthode est de choisir un échantillon de trois éléments, et prendre la médiane de cet échantillon comme pivot. C'est le quicksort médiane de 3, qui est étudié dans [Knu98]. Le dernier quicksort présenté est le quicksort médiane 3-3, qui sélectionne une pseudo-médiane de 9 éléments comme suit. On choisit trois échantillons de trois éléments chacun, la médiane de chaque échantillon est choisie pour créer un nouvel échantillon, dont la médiane est sélectionnée comme pivot. Cette technique est appelée pseudo-médiane car le pivot peut être le quatrième, le cinquième ou le sixième des neufs éléments. Sélectionner la pseudo-médiane coûte en moyenne $32/3$ comparaisons, ce qui est bien moins que le coût de détermination de la médiane qui est de 14 comparaisons.

L'algorithme optimisé de Bentley et McIlroy est alors :

Quicksort optimisé de Bentley et McIlroy
- Pour un tableau de taille < 7, utiliser le quicksort de base
- Pour un tableau de taille ≥ 7 et ≤ 39, utiliser quicksort médiane de 3
- Pour un tableau de taille > 40 utiliser quicksort médiane 3-3

Et ceci de façon récursive

L'analyse du quicksort de Bentley-McIlroy

Pour l'analyse de cet algorithme, on suppose que le tableau à trier est rempli par des éléments distincts, et que la distribution de probabilité sur les permutations possibles est uniforme. La manière dont on sélectionne les échantillons est alors sans importance. En effet il est équivalent de considérer que le tableau est rempli par une permutation quelconque ou que les éléments des échantillons sont choisis au hasard, de façon uniforme. Dans les deux formalisations, tous les éléments ont la même probabilité d'être choisi dans un échantillon.

Notons f_n le nombre moyen de comparaisons effectué pour trier un tableau de taille n par l'algorithme de quicksort optimisé présenté précédemment. De par la construction récursive de l'algorithme, on peut écrire une récurrence satisfaite par la suite f_n,

$$f_n = t_n + \sum_k \pi_k (f_{k-1} + f_{n-k}).$$

ou t_n désigne le nombre de comparaisons faites lors de la première étape de l'algorithme : la séparation en sous-tableaux, avant les appels récursifs et π_k la probabilité que le pivot choisi soit le k-ième élément du tableau. Après calcul de ces valeurs on obtient la récurrence satisfaite par f_n. On note

$(x)_k := x(x-1)\dots(x-k-1)$.

$$f_n = n-1+32/3+\frac{(3!)^2}{(n)_9}\sum_{k=4}^{n-5}6(k-1)_3(n-k)_5 f_{k-1}+\frac{(3!)^2}{(n)_9}\sum_{k=5}^{n-4}20(k-1)_4(n-k)_4 f_{k-1}+$$
$$\frac{(3!)^2}{(n)_9}\sum_{k=6}^{n-3}6(k-1)_5(n-k)_3 f_{k-1} \qquad n>40. \tag{1.1}$$

Cette récurrence se traduit en une équation intégrale lorsqu'on multiplie chaque membre de l'équation par z^n, puis qu'on somme sur n. Le membre de gauche devient $f(z)$, et les termes de droite deviennent des expressions intégrales fonctions de f. En dérivant cette relation, on obtient l'équation différentielle du lemme suivant.

Lemme 1.1 *L'équation différentielle satisfaite par la série génératrice $f(z)=\sum_n f_n z^n$ est*

$$(1-z)^9 f^{(9)}(z) = (1-z)^9 t(z) + 25920(1-z)^3 f^{(3)}(z) + 17280(1-z)^4 f^{(4)}(z) + 1296(1-z)^5 f^{(5)}(z),$$

où $t(z)=\frac{10!}{(1-z)^{11}}+\frac{26}{3}\frac{9!}{(1-z)^{10}}$, et $f^{(i)}$ vaut $\frac{\partial^i f(z)}{\partial z^i}$.

Cette équation différentielle est obtenue à partir de l'algorithme quicksort médiane 3-3, qui n'est en pratique utilisé qu'à partir de $n=40$. On considère alors les 49 premières valeurs de f_n, ou de façon équivalente les quarante-neuf premières dérivées de f évaluées en 0, comme des conditions initiales. L'équation différentielle vérifiée par f est une équation d'Euler et peut donc être résolue de manière explicite en fonction des racines α_i du polynôme indiciel qui est ici $I(x)=(x-2)x(x+1)(x+2)(x^5+35x^4+515x^3+4165x^2+19188x+36576)$ (voir [Dur00, Dur03]). La solution générale de cette équation différentielle est

$$f(z)=\frac{10!}{2311776}\frac{1}{(1-z)^2}\log_e\frac{1}{1-z}-\frac{26/3}{1-z}+\sum_i\frac{\lambda_{\alpha_i}}{(1-z)^{\alpha_i}}.$$

La connaissance des conditions initiales permet de déterminer les coefficients λ_{α_i}, et ainsi d'atteindre une connaissance complète de $f(z)$. On se contente ici d'écrire les premiers termes du développement de $f(z)$ au voisinage de sa singularité dominante 1. Les termes suivants sont tous accessibles, mais ne présentent qu'un intérêt secondaire.

$$f(z)\sim_{z=1}\frac{12600}{8027}\frac{1}{(1-z)^2}\ln\frac{1}{1-z}+\frac{\lambda_2}{(1-z)^2}-\frac{26/3}{1-z}.$$

La valeur λ_2 est égale à la fraction rationnelle N/D où $N=29959837331155094190991 57$ et $D=80382815077966511077422 00$, ce qui s'approxime par $\lambda_2=0.3727\dots$.

La connaissance de cette série génératrice permet d'obtenir les coefficients f_n, en utilisant les résultats de la section 1.1.2. On obtient alors le théorème

Théorème 1.3 *Le développement asymptotique de f_n, la complexité moyenne du quicksort optimisé de Bentley et McIlroy, est*

$$f_n=\frac{12600}{8027}n\log_e n+\left((\gamma-1)\frac{12600}{8027}-\lambda_2\right)n+\frac{12600}{8027}\log_e n+\left(\left(\frac{1}{2}+\gamma\right)\frac{12600}{8027}-\frac{26}{3}-\lambda_2\right)+O\left(\frac{1}{n}\right).$$

Comme $\lambda_2 = 0.3727\ldots$ et que γ est la constante d'Euler, le développement avec des constantes approchées numériquement est

$$f_n = 1.5697\, n \log_e n - 1.0363\, n + 1.5697 \log_e n - 7.3484 + O(1/n).$$

Ceci est très proche de ce que Bentley et McIlroy ont trouvé expérimentalement, soit une complexité égale à $f_n = 1.5783\, n \log_e n - 0.74\, n$, obtenu par une régression par moindres carrés.

Optimisations

A partir de l'idée de Bentley et McIlroy de créer un quicksort composé des trois quicksorts, classique, médiane de 3 et médiane 3-3, on va ici chercher les valeurs de seuil idéales dictant les changements d'algorithme, qui optimisent le nombre moyen de comparaisons. Celles de l'algorithme présenté par Bentley et McIlroy ont été déterminées par ces auteurs de façon expérimentale. Les valeurs de seuil optimales ne dépendent que des récurrence respectives des complexités moyenne des trois quicksorts utilisés.

On propose ici une amélioration du quicksort de Bentley et McIlroy, fondée sur les choix

$n < 15$	quicksort classique
$15 \leq n < 86$	quicksort médiane de 3
$86 \leq n$	quicksort médiane 3-3

On obtient alors un nombre moyen de comparaisons pour un tableau de taille n qui est

$$c_n = 1.5697\, n \log_e n - 1.1512\, n + 1.5697 \log_e n - 7.4633 + o(1),$$

qui est légèrement inférieur à f_n. Pour un tableau de taille 1000, le quicksort "optimal" ci-dessus est 1.2% meilleur que celui de Bentley et McIlroy, et 12% meilleur que le quicksort classique. Cependant il faut garder à l'esprit que d'autres paramètres comptent dans le temps d'exécution de l'algorithme, comme le nombre d'échanges, qu'on ne prend pas en compte ici. On remarque que la complexité moyenne c_n est proche de la limite inférieure théorique de la moyenne qui est $\log_e n! \sim 1.4426\, n \log_e n - 1.44\, n$.

Le quicksort présenté dans cette section est optimal dans sa catégorie, mais de nombreuses autres stratégies de choix de pivot sont possibles. Par exemple prendre un échantillon de taille 3^k, déterminer 3^{k-1} médianes de triplets, puis 3^{k-2} médianes de médianes, et ce jusqu'à obtention du pivot ; ou bien choisir m pivots, $m > 1$, et ensuite diviser le tableau en $m + 1$ sous tableaux ; ou encore sélectionner m pivots parmi un échantillon de taille plus grande que m. L'analyse en moyenne ou en distribution de ces variantes fait très souvent intervenir des équations d'Euler et est donc du ressort des méthodes que nous venons d'exposer.

Chapitre 2

Asymptotique bivariée et listes à sauts

Sommaire

Ce chapitre est consacré à l'analyse asymptotique de coefficients de séries génératrices multivariées. On considère des séries génératrices à singularités confluentes du type $(1-z)^{-a}(1-zu)^{-b}(1-u)^{-c}$. Ces résultats sont appliqués à plusieurs problèmes, dont notamment les listes à sauts.

Résultats principaux. Le théorème 2.3 (p. 60) énonce l'estimation asymptotique des coefficients bivariés de la série génératrice (2.1), soit

$$[z^n u^k]\left(\frac{1}{1-z}\right)^a\left(\frac{1}{1-zu}\right)^b\left(\frac{1}{1-u}\right)^c \sim \frac{k^{b+c-1}n^{a-1}}{\Gamma(a)}\frac{1}{\Gamma(b+c)}{}_2F_1\left(1-a,b;b+c;\frac{k}{n}\right),$$

lorsque $\Re(a)>0$, $\Re(b)>0$ et $\Re(c)>0$.

Le théorème 2.1 (p. 51) et les propositions 2.9, 2.10 2.11, 2.12, et 2.13 énoncent toutes les propriétés des listes à sauts, qui sont rappelées ici.

- Il y a C_n listes à sauts de taille n, où C_n désigne les nombres de Catalan.
- Le coût moyen d'insertion dans une liste à sauts est $\sim 2\log n$.

  La profondeur d'un élément est définie comme le nombre de pointeurs qu'il faut suivre pour accéder à cet élément.
- La longueur de cheminement interne moyenne, égale à la somme des profondeurs des nœuds est équivalente à $2n\log n$.
- Le profil d'une liste à sauts, défini comme la distribution de la profondeur de ses éléments, est en moyenne asymptotiquement gaussien.
- La hauteur d'une liste à sauts, qui est le maximum des profondeurs des nœuds, est en moyenne inférieure à la hauteur moyenne d'un arbre binaire de recherche, soit $c\log n$, où $c = 4.31\ldots$.

– La profondeur moyenne du k-ième élément dans une liste à sauts de taille n est équivalente à $\log k + \log(n-k)$, pour $\log^2 n \leq k \leq n - \log^2 n$.

2.1 Introduction

Ce chapitre traite d'analyse bivariée de séries génératrices à singularités confluentes. Il s'agit d'obtenir des résultats asymptotiques pour des coefficients de séries génératrices bivariées du type de l'équation (2.1), lesquelles interviennent à de nombreuses reprises en analyse d'algorithme. Des exemples d'application sont présentés.

La section 2.2 de ce chapitre est consacrée à la mise au point d'outils d'analyse bivariée pour le cas plus simple des séries génératrices de la forme $(1-z)^{-a}(1-zu)^{-b}$. On utilisera la particularité du problème, qui est que la série possède deux singularités confluentes en la variable z, mais n'a qu'une unique singularité en la variable u. La méthode consiste alors à contourner la difficulté de la confluence, pour d'abord extraire le coefficient de u^k, et se ramener ainsi à un problème d'analyse univariée qu'on sait résoudre.

Le cas général est l'objet d'une section ultérieure. Il s'agit de déterminer la nature asymptotique des coefficients bivariés de séries génératrices possédant plusieurs singularités confluentes. On s'intéresse en particulier à des séries du type

$$\left(\frac{1}{1-z}\right)^a \left(\frac{1}{1-u}\right)^c \left(\frac{1}{1-zu}\right)^b, \tag{2.1}$$

avec éventuellement des facteurs logarithmiques supplémentaires. Les deux variables u et z ont des rôles symétriques, et il est indispensable de gérer la confluence de singularités. Les sections 2.5 et 2.6 étudient cette question. Le théorème 2.3 énonce l'asymptotique des coefficients de la série (2.1). Ce résultat est ensuite généralisé à une famille de fonctions équivalentes. La preuve utilise la transformation de Mellin pour "séparer" les singularités, et ainsi obtenir le coefficient de u^k de la série de l'équation (2.1) comme une série hypergéométrique en z. Cette série peut ensuite être utilisée pour obtenir le coefficient de z^n. Le résultat final met en jeu une série hypergéométrique en l'argument k/n, avec a, b et c comme paramètres.

Un premier exemple d'application de ces résultats est l'analyse des listes à sauts. Les listes à sauts, introduites par Frédéric Cazals et Hervé Brönnimann, sont une structure de donnée basées sur des listes chaînées ordonnées, et inspirées des skiplists dues à Pugh [Pug90]. Les algorithmes qui permettent d'utiliser cette structure sont présentés (Recherche - Insertion - Suppression), ainsi qu'une analyse complète des coûts moyens des paramètres principaux. Cette structure doit être comparée aux arbres binaires de recherche, et l'analyse montre que les ordres de grandeur des coûts sont identiques. Elle présente des propriétés intéressantes, comme une recherche d'intervalles très rapide, ou encore un accès en temps constant aux plus petits éléments de l'ensemble. L'analyse de la profondeur moyenne du k-ième élément dans une liste à saut donne lieu à des séries génératrices à singularités confluentes. C'est cette analyse, techniquement délicate, qui m'a conduite à étudier de manière générale les questions d'asymptotiques traitées dans ce chapitre.

Les séries génératrices de la forme (2.1) apparaissent dans d'autres parties de l'algorithmique. On traite l'exemple des algorithmes de sélection rapide (quickselect) et de tri rapide partiel (partial quicksort), ce dernier étant dû à Martinez [Mar04], dont l'analyse fait apparaître des séries génératrices présentant ce type de singularités. On examine également dans la section 2.7 l'exemple de la fréquence d'apparition d'un symbole dans une chaîne de caractères issue d'une source à deux composantes primitives. Dans le cas où les deux composantes ont des contributions du même ordre

de grandeur, une série génératrice possédant une confluence de singularité intervient, ce qui entraîne pour la fréquence étudiée une distribution uniforme sur un intervalle. Cet exemple permet de retrouver un résultat de De Falco, Goldwurm et Lonati [DGL03], originellement établi par des méthodes d'analyse matricielle.

Les méthodes d'asymptotique bivariée développées ici pour le cas particulier de séries de type (2.1) s'inscrivent dans une problématique plus générale d'asymptotique bivariée, qui s'avère nettement plus délicate que les problèmes univariés correspondant, de par les interférences possibles entre les deux variables. Robin Pemantle et Mark Wilson étudient la question d'asymptotique multivariée de façon générale selon la géométrie des singularités dans [PW97, PW03]. Les cas étudiés dans ce chapitre ne sont pas couverts par leurs résultats.

Ce chapitre est divisé en sept sections. La première section 2.2 présente un cas particulier d'analyse asymptotique bivarié, suffisant pour l'application des listes à sauts. La section 2.3 est consacrée à l'étude des arbres binaires de recherche, auxquels on veut comparer les listes à sauts. La section (2.4) présente et étudie les listes à sauts. Les section 2.5 et 2.6 contiennent des résultats généraux d'asymptotique bivariée, et leur preuve. Finalement les deux dernières sections sont consacrées à des exemples d'applications.

2.2 Cas d'une unique confluence

On commence par traiter le cas d'une confluence unique. Cela permet de rappeler les méthodes d'analyse de singularités, et de montrer les difficultés d'extension des méthodes univariées à un cadre bivarié. De plus ce cas particulier est utile pour l'analyse des listes à sauts.

Dans le cas où $c = 0$ dans l'équation (2.1), il n'y a qu'une confluence de singularité, et le problème se simplifie alors en l'étude de la fonction

$$\left(\frac{1}{1-z}\right)^a \left(\frac{1}{1-zu}\right)^b. \tag{2.2}$$

Cette section est consacrée à l'étude de la fonction définie dans l'équation (2.2). Cette fonction possède deux singularités, situées en $z = 1$ et $z = u^{-1}$, qui se rejoignent lorsque u tend vers 1. Il y a alors une *confluence de singularités* et c'est le comportement de la fonction au voisinage du point de confluence $(z, u) = (1, 1)$ qui dicte le comportement asymptotique des coefficients.

On remarque que selon la variable u, il n'y a qu'une singularité située en $u = 1/z$. Ce problème d'extraction bivariée à singularités confluentes peut donc, dans ce cas particulier, être réduit à un problème d'extraction où une seule singularité intervient, puis à un problème univarié. Il est alors possible de déterminer l'asymptotique des coefficients de cette série génératrice dans un domaine raisonnable par calcul d'une intégrale de Cauchy selon un double contour de Hankel.

On rappelle tout d'abord la définition et l'usage d'un contour de Hankel en dimension 1 pour l'analyse de singularités, ce qui est un résumé de Flajolet et Sedgewick [FS05], et dans la deuxième partie, on généralise ces résultats en suivant les méthodes présentes dans l'article de Drmota [Drm94a] pour obtenir le résultat recherché grâce à une intégration selon un double contour de Hankel.

2.2.1 Contour de Hankel en dimension 1

L'intégration par contour de Hankel permet de traduire l'information dont on dispose sur la fonction au voisinage de ses singularités en information sur l'asymptotique des coefficients de son développement de Taylor. Pour une présentation détaillée, voir [FS05]. Le module de la plus petite

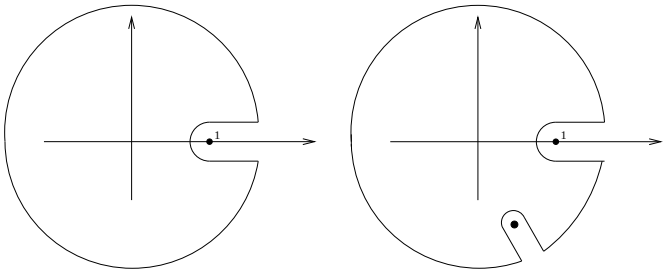

FIG. 2.1 – Contour de Hankel pour une singularité [gauche] ou deux singularités [droite].

singularité donne le comportement exponentiel des coefficients, et le comportement de la fonction au voisinage de la singularité permet d'affiner cette première approximation.

Le contexte est le suivant : on dispose d'une fonction $f(z) = \sum f_n z^n$, et on veut déterminer l'asymptotique de la suite f_n. De par la combinatoire du problème, on a fréquemment accès à un équivalent de la série génératrice, comme on l'a montré dans le chapitre 1. Les hypothèses faites dans ce sens concernent donc un grand nombre de problèmes. Souvent, on dispose d'informations supplémentaires, comme par exemple que ces coefficients f_n sont positifs. Cela est extrêmement fréquent en combinatoire, car la quantité f_n correspond typiquement à un dénombrement de structure, ou encore à une probabilité.

On s'intéresse aux fonctions qui n'ont qu'une singularité de module minimal située en $z = 1$ et qui satisfont $g(z) \sim \left(\frac{1}{1-z}\right)^a$ dans un voisinage suffisamment grand de $z = 1$. Si cette singularité est située en un point $\rho \neq 1$, on peut se ramener au cas précédent par une simple homothétie ($z \to z/\rho$). Pour la clarté de l'exposé on étudie d'abord la fonction $f(z) = \left(\frac{1}{1-z}\right)^a$, puis on étend les résultats obtenus aux fonctions équivalentes à f.

Concernant l'asymptotique des coefficients de la fonction f, on énonce la proposition suivante :

Proposition 2.1 *La fonction* $f(z) = \left(\frac{1}{1-z}\right)^a$, $-a \notin \mathbb{N}$ *satisfait*

$$[z^n]f(z) \sim \frac{n^{a-1}}{\Gamma(a)}.$$

La condition $-a \notin \mathbb{N}$ signifie que $f(z)$ n'est pas un polynôme. Cette condition est toujours imposée dans la suite.

Remarque. Le développement asymptotique des coefficients de la série génératrice peut être poussé à n'importe quel ordre. On se restreint ici volontairement à l'ordre 1, d'une part pour la clarté de la présentation, et d'autre part parce que les applications présentes dans la suite du chapitre n'utilisent que le premier ordre.

Preuve. Le coefficient f_n s'exprime de façon intégrale par le théorème de Cauchy,

$$f_n = \frac{1}{2i\pi} \int_\Gamma \left(\frac{1}{1-z}\right)^a \frac{dz}{z^{n+1}},$$

où Γ désigne un contour autour de 0. On choisit le chemin d'intégration comme sur la Figure 2.1 de façon à entourer la singularité. On pose $\Gamma = \Gamma_1 + \Gamma_2 + \Gamma_3 + \Gamma_4$, où

$$\begin{cases} \Gamma_1 = \{z = 1 + \frac{e^{i\theta}}{n}, \theta \in [\pi/2, 3\pi/2]\} \\ \Gamma_2 = \{z = x + \frac{i}{n}, x \in [1, R]\} \\ \Gamma_3 = \{z = x - \frac{i}{n}, x \in [1, R]\} \\ \Gamma_4 = \{z = Re^{i\theta}, \Re(z) \leq R\}. \end{cases}$$

La borne R peut par exemple être choisie comme étant égale à $1+\frac{\log^2 n}{n}$. Cette valeur doit être prise suffisamment grande pour permettre de majorer l'intégrale de reste le long du contour Γ_4.

On utilise ensuite le changement de variable $z = 1 + \frac{t}{n}$, ce qui permet de se focaliser sur le voisinage de la singularité dans la bonne échelle. On obtient alors l'égalité suivante

$$f_n = \frac{1}{n2i\pi}\int_{\mathcal{C}'} \frac{n^a}{(-t)^a}\frac{dt}{(1+\frac{t}{n})^{n+1}}.$$

Le chemin $\mathcal{C}'$ correspond à la transformation du chemin Γ selon le changement de variable. Les notations analogues sont utilisées pour les chemins Γ_i. On sépare l'intégrale en 2, tout d'abord, $\mathcal{C}'_1+\mathcal{C}'_2+\mathcal{C}'_3$ qui est la partie principale, et ensuite $\mathcal{C}'_4$ qui fournit une contribution négligeable.

La valeur de R étant fixée à $1+\frac{\log^2 n}{n}$, le long du chemin $\mathcal{C}'_1+\mathcal{C}'_2+\mathcal{C}'_3$, la quantité t est non bornée lorsque n tend vers l'infini, tandis que $1+t/n$ reste au voisinage de 1. On a donc l'estimation $(1+\frac{t}{n})^{-n-1} = e^{-t}\left(1+O(\log^4 n/n)\right)$. Cela permet d'écrire

$$\frac{1}{n2i\pi}\int_{\mathcal{C}'_1+\mathcal{C}'_2+\mathcal{C}'_3} \frac{n^a}{(-t)^a}\frac{dt}{(1+\frac{t}{n})^{n+1}} \sim \frac{n^{a-1}}{2i\pi}\int_{\mathcal{C}'_1+\mathcal{C}'_2+\mathcal{C}'_3}\frac{e^{-t}}{(-t)^a}\,dt.$$

Le membre droit de l'équation précédente est évaluée grâce au lemme suivant.

Lemme 2.1 *Soit $\mathcal{D}$ le chemin qui part de $\infty+i$, tourne autour de l'origine dans le sens direct et termine en $\infty-i$. Alors*

$$\frac{1}{2i\pi}\int_{\mathcal{D}}\frac{e^{-t}}{(-t)^a}\,dt = \frac{1}{\Gamma(a)}.$$

La preuve de ce lemme repose sur la définition de la fonction Γ et la formule des compléments qui peut être trouvée dans [AS73], ainsi que sur une technique classique originellement due à Hankel. La preuve du lemme est donnée dans [FS05].

L'intégrale qui nous intéresse est alors décomposée en $\int_{\mathcal{C}'_1+\mathcal{C}'_2+\mathcal{C}'_3} = \int_{\mathcal{D}} - \int_{\mathcal{R}}$. Sur ce chemin $\mathcal{R}$, l'intégrande $\frac{e^{-t}}{(-t)^a}$ est à décroissance exponentielle. En choisissant la valeur $R = 1+\frac{\log^2 n}{n}$, on obtient la borne $\int_R \frac{e^{-t}}{(-t)^a}\,dt = O(e^{-\log^2 n})$. Ceci permet d'écrire que $\int_{\mathcal{C}'_1+\mathcal{C}'_2+\mathcal{C}'_3}\frac{e^{-t}}{(-t)^a}\,dt = \frac{1}{\Gamma(a)} + O(e^{-\log^2 n})$. Le résultat peut alors être énoncé comme un corollaire du lemme 2.1.

Corollaire 2.1 *Soit $\mathcal{D}_n$ le chemin qui part de $\log^2 n+i$, tourne autour de l'origine dans le sens direct et termine en $\log^2 n - i$. Alors*

$$\frac{1}{2i\pi}\int_{\mathcal{D}_n}\frac{e^{-t}}{(-t)^a}\,dt = \frac{1}{\Gamma(a)}\left(1+O(e^{-\log^2 n})\right).$$

Pour la partie de l'intégrale le long de Γ_4, le changement de variable n'est pas nécessaire. La fonction $\frac{1}{(1-z)^a}$ est bornée supérieurement sur Γ_4 par n^a, et le dénominateur $\frac{1}{z^{n+1}}$ est borné par R^{-n}. Finalement, la contribution de cette partie de l'intégrale est exponentiellement petite lorsque n tend vers l'infini.

On peut alors conclure

$$f_n \sim \frac{n^{a-1}}{\Gamma(a)}.$$

■

Cette proposition se généralise au cas d'une fonction g qui est équivalente à la fonction $f(z) = \frac{1}{(1-z)^a}$ au voisinage de la singularité située ici en 1.

Proposition 2.2 *Si la fonction g est Δ-analytique (voir Définition 1.1) et si dans un voisinage de 1 on a $g(z) \sim \frac{1}{(1-z)^a}$, avec $-a \notin \mathbb{N}$, alors les coefficients de la série g vérifient*

$$[z^n]g(z) \sim \frac{n^{a-1}}{\Gamma(a)}.$$

La preuve précédente s'adapte sans difficulté à cette généralisation. Pour une étude complète, on peut se reporter à [FS05].

Remarque. Si la série génératrice étudiée a plusieurs singularités isolées de module minimal, la forme asymptotique des coefficients s'obtient en sommant les contributions individuelles de chaque singularité. Cela s'établit en construisant un contour d'intégration qui entoure toutes les singularités, comme présenté sur la figure 2.1.

Remarque. Il arrive fréquemment que les fonctions de bases auxquelles on cherche à se comparer ne soient pas seulement du type simple $(1-z)^{-a}$, mais également de la forme $(1-z)^{-a}\log^k(1/(1-z))$. L'asymptotique des coefficients s'obtient en généralisant la preuve de la proposition 2.1, et on obtient alors si $-a \notin \mathbb{N}$

$$[z^n]\left(\frac{1}{1-z}\right)^a \log^k\left(\frac{1}{1-z}\right) \sim \frac{n^{a-1}}{\Gamma(a)}\log^k n.$$

Ce résultat est également valable pour une fonction f équivalente à $(1-z)^{-a}\log^k(1/(1-z))$. On a alors $[z^n]f(z) \sim \frac{n^{a-1}}{\Gamma(a)}\log^k n$. Un énoncé complet peut être trouvé dans [FS05].

2.2.2 Contour de Hankel en deux variables

La généralisation de la section précédente au cas bivarié n'est pas immédiate. Je présente dans cette section un cas particulier qui est ensuite utilisé dans la section 2.4 pour l'analyse des listes à sauts [BCD03]. Cette méthode de double contour de Hankel est inspirée des travaux de Drmota [Drm94c].

On s'intéresse ici à la fonction bivariée $f(z,u) = \frac{1}{(1-z)^a}\frac{1}{(1-zu)^b}$, et on suppose que a et b ne sont pas des entiers négatifs. Cette fonction est un exemple qui se veut représentatif, car de nombreuses fonctions s'y ramènent. De plus, comme dans la section précédente, les résultats obtenus sur cette fonction se généralisent aux fonctions qui lui sont équivalentes.

L'objectif est d'extraire le coefficient de $[z^n u^k]$ de la fonction f. Pour cela, on énonce la proposition suivante.

Proposition 2.3 *Si k est compris entre $\log^2 n$ et $n - \log^2 n$ alors, lorsque n tend vers l'infini, on a uniformément en k*

$$[z^n][u^k]f(z,u) \sim \frac{k^{b-1}}{\Gamma(b)}\frac{(n-k)^{a-1}}{\Gamma(a)}.$$

Preuve. La formule de Cauchy fournit l'expression de $f_{n,k} = [z^n][u^k]f(z,u)$:

$$f_{n,k} = \frac{1}{(2i\pi)^2}\int_{0+}\int_{0+}\frac{1}{(1-z)^a}\frac{1}{(1-zu)^b}\frac{du}{u^{k+1}}\frac{dz}{z^{n+1}}. \tag{2.3}$$

Les contours d'intégration notés $0+$ sont des contours de Hankel autour de l'origine qui sont détaillés ici.

Contour pour z : Ce contour, orienté dans le sens direct, est l'union de 4 chemins $\Gamma_1, \ldots \Gamma_4$,

$$\begin{aligned}
\Gamma_1 &= \{z = (1 + t/n) \mid \Re t < 0, |t| = 1\} \\
\Gamma_2 &= \{z = (1 + t/n) \mid 0 \leq \Re t \leq \log^2 n, \Im t = 1\} \\
\Gamma_3 &= \{z = (1 + t/n) \mid 0 \leq \Re t \leq \log^2 n, \Im t = -1\} \\
\Gamma_4 &= \left\{ z \mid |z| = \left| 1 + \frac{\log^2 n + i}{n} \right| \Re(z) \leq 1 + \frac{\log^2 n}{n} \right\}.
\end{aligned}$$

Contour pour u : Ce contour, orienté dans le sens direct, est l'union de 4 chemins $\Delta_1, \ldots, \Delta_4$, qui dépendent de z (on intègre d'abord par rapport à u, puis par rapport à z).

$$\Delta_1 = \{u = (1 + h/k) \mid \Re h < -R(t), |h + R(t)| = 1\}$$

Le demi-cercle est décalé vers la gauche, de $R(t) = \max\{0, \Re(tk/n)\}$

$$\begin{aligned}
\Delta_2 &= \{u = (1 + h/k) \mid -R(t) \leq \Re h \leq M, \Im h = 1\} \\
\Delta_3 &= \{u = (1 + h/k) \mid -R(t) \leq \Re h \leq M, \Im h = -1\} \\
\Delta_4 &= \left\{ u \mid |u| = \left| 1 + \frac{M + i}{k} \right| \Re(u) < 1 + \frac{M}{k} \right\}
\end{aligned}$$

La valeur M est choisie comme étant égale à $\max(\log^2 k, \log^{3/2} n)$, ce qui permet de garantir un contour qui va suffisamment loin dans le plan des u, même pour les petites valeurs de k.

Une fois les contours d'intégration bien définis, le calcul de l'intégrale double reprend les méthodes utilisées pour le cas univarié. Il s'agit d'extraire la partie principale, puis de borner les termes d'erreur, sachant qu'on intégrera d'abord par rapport à u, puis par rapport à z. On désigne par $\Delta_{i,j}$ l'union des deux chemins Δ_i et Δ_j, et l'on étend cette notation à l'union de plus de deux chemins ainsi qu'au contour Γ. Comme dans le cas univarié, le chemin d'intégration apportant la contribution principale est la zone autour des singularités. Les autres parties sont traitées comme des termes d'erreurs. L'intégrale double est découpée en trois morceaux. Tout d'abord, on s'intéresse à la partie principale, qui est $\Delta_{1,2,3} \times \Gamma_{1,2,3}$, puis on étudie les termes d'erreur en deux étapes, $\Delta_{1,2,3,4} \times \Gamma_4$ puis $\Delta_4 \times \Gamma_{1,2,3,4}$.

Pour la partie principale, le processus est très similaire au cas d'une seule variable. On applique les changements de variables $z = 1 + t/n$ et $u = 1 + h/k$. Les termes z^{-n-1} et u^{-k-1} peuvent alors être remplacés par e^{-t} et e^{-h}. On obtient alors

$$\int_{\Gamma_{1,2,3}} \int_{\Delta_{1,2,3}} \frac{1}{(1-z)^a} \frac{1}{(1-zu)^b} \frac{du}{u^{k+1}} \frac{dz}{z^{n+1}} \sim \frac{1}{nk} \int e^{-t} \left(\frac{-n}{t} \right)^a \int \left(\frac{-k}{h + t(k+h)/n} \right)^b e^{-h} dh dt.$$

Les contours d'intégration du membre droit sont déduits de Γ et Δ par homothétie et ne sont pas explicités. Dans la suite on note Δ'_i et Γ'_i les chemins d'intégration pour h et t qui correspondent aux chemins Δ_i et Γ_i pour u et z.

Partie principale

On vise ici à approximer la contribution principale, notée P, qui est donnée par l'intégrale qui prend en compte l'excroissance $\Gamma_{1,2,3}$ du contour d'intégration selon z, et $\Delta_{1,2,3}$, celle du contour selon u. Dans cette zone, on peut faire l'approximation $k + h = k(1 + O(M/k))$, ce qui permet d'écrire

$$P = \frac{1}{nk(2i\pi)^2} \int_{\Gamma'_{1,2,3}} e^{-t} \left(\frac{-n}{t} \right)^a \int_{\Delta'_{1,2,3}} \frac{(-k)^b}{(h + tk/n)^b} e^{-h} dh dt \left(1 + O\left(\frac{M}{k} \right) \right). \tag{2.4}$$

On intègre tout d'abord par rapport à h selon le contour $\Delta'_{1,2,3}$. Pour cela, on effectue le changement de variable $h' = h + \frac{tk}{n}$, ce qui transforme l'intégrale $\int_{\Delta'_{1,2,3}} \frac{(-1)^b}{(h+tk/n)^b} e^{-h} dh$ en $e^{\frac{tk}{n}} \int \frac{1}{(-h')^b} e^{-h'} dh'$. Le contour d'intégration de cette nouvelle intégrale part de $i + M + \frac{tk}{n}$, tourne autour de $-R(t) + \frac{tk}{n}$ dans le sens direct et termine en $-i + M + \frac{tk}{n}$. C'est maintenant que le décalage du contour de Hankel Δ par $R(t)$ devient utile, car par définition, $-R(t) + \frac{tk}{n}$ est inférieur ou égal à zéro. De la sorte ce contour d'intégration enclot 0. Le corollaire 2.1 s'applique alors, et on a l'égalité $\frac{1}{2i\pi} \int \frac{1}{(-h')^b} e^{-h'} dh' = \frac{1}{\Gamma(b)} \left(1 + O\left(e^{-M}\right)\right)$. La valeur du terme d'erreur est obtenue en observant que $\Re(i + M + \frac{tk}{n})$ est supérieur à $M - 1$, car sur les chemins considérés la partie réelle de t est supérieure à -1. Ceci donne l'estimation

$$\frac{1}{2i\pi} \int_{\Delta'_{1,2,3}} \left(\frac{-1}{h + tk/n}\right)^b e^{-h-tk/n} dh = \frac{1}{\Gamma(b)} \left(1 + O\left(e^{-M}\right)\right).$$

L'équation (2.4) peut alors se récrire :

$$P = \frac{1}{2i\pi} \frac{n^{a-1} k^{b-1}}{\Gamma(b)} \int_{\Gamma' 1,2,3} e^{-t} \frac{1}{(-t)^a} e^{tk/n} dt \left(1 + O\left(\frac{M}{k}\right)\right).$$

Un dernier changement de variable $t' = t(1 - k/n)$ (inspiré par le terme $e^{-t+tk/n}$ de l'intégrale) transforme P en $\frac{1}{2i\pi} \int e^{-t'} \frac{1}{(-t')^a} dt'$, à un facteur $\frac{n^{a-1}(1-k/n)^{a-1}}{\Gamma(a)} \frac{k^{b-1}}{\Gamma(b)}$ près, modulo les termes d'erreur qui restent inchangés. Le contour d'intégration part de $\log^2 n(1 - k/n) + i(1 - k/n)$, contourne l'origine et se termine en $\log^2 n(1 - k/n) - i(1 - k/n)$. Si $k/n < a < 1$, l'extrémité du contour $\log^2 n(1 - k/n)$ est asymptotiquement supérieure à M. On applique alors le corollaire 2.1, ce qui donne

$$P = \frac{(n-k)^{a-1}}{\Gamma(a)} \frac{k^{b-1}}{\Gamma(b)} + \left(1 + O\left(\frac{M}{k}\right)\right).$$

Termes d'erreur

Les autres composantes de l'intégrale vont seulement contribuer en tant que termes d'erreur. Ils seront bornés en deux temps, tout d'abord la partie $\Gamma_{1,2,3} \times \Delta_4$, puis $\Gamma_4 \times \Delta_{1,2,3,4}$.

- Pour la première partie, on doit borner l'intégrale

$$\frac{1}{(2i\pi)^2} \int_{\Gamma_{1,2,3}} \int_{\Delta_4} \frac{1}{(1-z)^a} \frac{1}{(1-zu)^b} \frac{du}{u^{k+1}} \frac{dz}{z^{n+1}}. \tag{2.5}$$

Sur ces contours, on dispose des majorations suivantes : $\left|\frac{1}{1-z}\right| \leq n$ et $\left|\frac{1}{1-zu}\right| \leq \frac{k}{M}$ car $|h| \geq M$. Le terme z^{-n-1} peut être borné par une constante, et ne pose pas de problème (car z se trouve sur le contour $\Gamma_{1,2,3}$), et le terme u^{-k-1} est celui qui permet de borner l'intégrale car on a $|u^{-k-1}| \leq \left(1 + \frac{M}{k}\right)^{-k-1}$. Cette majoration permet d'obtenir la borne $u^{-k-1} = O\left(e^{-M}\right)$. La compilation de tous ces résultats, en supposant $k \geq \log^2 n$, montre que l'intégrale de l'équation (2.5) est $O\left(n^a k^b e^{-M}\right)$ et est donc bien un terme d'erreur par rapport au terme principal, ceci car M est supérieur à $\log^{3/2} n$.

- La deuxième partie correspond à l'intégrale

$$\frac{1}{(2i\pi)^2} \int_{\Gamma_4} \int_{\Delta_{1,2,3,4}} \frac{1}{(1-z)^a} \frac{1}{(1-zu)^b} \frac{du}{u^{k+1}} \frac{dz}{z^{n+1}}. \tag{2.6}$$

Cette partie est plus facile à traiter, car c'est maintenant grâce au terme z^{-n-1} qu'on majore l'intégrale. Les majorations qui sont nécessaires ici sont : $\left|\frac{1}{1-z}\right| \leq \frac{1}{n}$, et $\left|\frac{1}{1-zu}\right| \leq \frac{1}{n}$. Le terme u^{-k-1} est toujours borné par une constante (cette majoration pourrait être améliorée, mais c'est sans intérêt ici) et enfin $|z^{-n-1}| = O(e^{-\log^2 n})$. Par conséquent l'intégrale de l'équation (2.6) est $O\left(n^a n^b e^{-\log^2 n}\right)$, ce qui représente également un terme d'erreur par rapport au terme principal.

Bilan

Finalement les développements qui précèdent prouvent que si k est compris entre $\log^2 n$ et αn, où α est une constante strictement inférieure à 1, on a $f_{n,k} \sim \frac{k^{b-1}}{\Gamma(b)} \frac{(n-k)^{a-1}}{\Gamma(a)}$. La proposition est en partie prouvée.

Symétrie. Pour terminer la preuve de la proposition 2.3, il faut élargir l'intervalle des valeurs de k. Pour cela, on effectue une symétrie selon k pour les coefficients $f_{n,k}$ et on examine à la série génératrice des coefficients $f_{n,n-k}$. On obtient cette série génératrice via la transformation qui à z associe zu et à u associe u^{-1}. On voit aisément que si on part de $f(z,u) = \sum f_{n,k} z^n u^k$ alors $f(zu, u^{-1}) = \sum f_{n,n-k} z^n u^k$. Or pour ce qui nous concerne, $f(z,u)$ est égale à $\frac{1}{(1-z)^a} \frac{1}{(1-zu)^b}$, et alors on a $f(zu, u^{-1}) = \frac{1}{(1-zu)^a} \frac{1}{(1-z)^b}$. Lorsqu'on applique la partie prouvée de la proposition à cette fonction, on obtient que $f_{n,n-k} \sim \frac{k^{b-1}}{\Gamma(b)} \frac{(n-k)^{a-1}}{\Gamma(a)}$ à condition que $n-k$ soit compris entre $\log^2 n$ et αn. Ceci complète bien l'intervalle pour k comme souhaité, et la proposition 2.3 est prouvée.

■

Généralisation

De manière similaire au cas univarié, ces résultats restent valables si les fonctions comportent des facteurs logarithmiques. La preuve qui est est omise conduit au résultat suivant :

Proposition 2.4 *La fonction* $f(z,u) = \left(\frac{1}{1-z}\right)^a \log^i\left(\frac{1}{1-z}\right)\left(\frac{1}{1-zu}\right)^b \log^j\left(\frac{1}{1-zu}\right)$ *satisfait*

$$[z^n u^k] f(z,u) \sim \frac{(n-k)^{a-1}}{\Gamma(a)} \log^i(n-k) \frac{k^{b-1}}{\Gamma(b)} \log^j k,$$

lorsque $k \in [\log^2 n, n - \log^2 n]$, *et que* n *tend vers l'infini.*

Remarque. Il est possible d'étendre légèrement le domaine de validité $[\log^2 n, n - \log^2 n]$ de l'énoncé précédent.

2.3 Arbres Binaires de Recherche

Cette section présente en détail la structure de donnée d'arbre binaire de recherche (ABR). Les paramètres classiques de cette structure sont la longueur de cheminement interne, la hauteur, la profondeur moyenne de chacun des éléments, dont on rappelle les moyennes. Ces rappels sont la plupart du temps accompagnées des preuves. Les arbres binaires de recherche sont extrêmement proches des listes à sauts (présentées section 2.4). Pour ces deux structures les mêmes paramètres sont analysés, et ensuite comparés. L'analyse est souvent basée sur les mêmes méthodes mais est

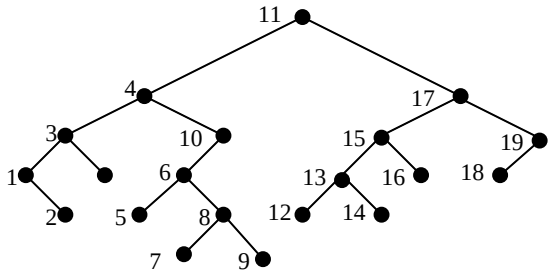

FIG. 2.2 – Un arbre binaire de recherche de taille 19

parfois beaucoup plus complexe pour les listes à sauts (par exemple la profondeur moyenne du k-ième élément). La comparaison est facilitée par le fait que les listes à sauts peuvent être vues comme des arbres binaires, comme on le montre dans la section 2.4.1.

2.3.1 Présentation

Définition 2.1 *Un arbre vide est un arbre binaire de recherche. Un arbre binaire de recherche non vide est un arbre binaire dont tous les sommets sont étiquetés et qui vérifie les deux propriétés suivantes.*
• *La valeur de la racine est supérieure à toutes les valeurs des sommets du sous-arbre gauche, et inférieure à toutes les valeurs des sommets du sous-arbre droit.*
• *Le sous-arbre droit et le sous-arbre gauche sont des arbres binaires de recherche.*

On peut consulter [CLRS01, Knu98, Mah92, Sed88, Vui80] pour une présentation générale de la structure de donnée et des algorithmes associés, ainsi que [Mah92, FS05] pour l'étude des différents paramètres comme longueur de cheminement interne, profil moyen, ou profondeur moyenne du k-ième élément. L'étude presque sûre de la profondeur et du profil est réalisée par Jabbour [Jab01] et Chauvin, Drmota et Jabbour dans [CDJH01]. L'étude de la hauteur des arbres binaires de recherche se trouve dans [Dev87, Drm01, Ree03].

On considère dans la suite des arbres binaires de recherches étiquetés de façon canonique par les entiers de 1 à n. Un arbre binaire de recherche est alors caractérisé de manière unique par sa forme. La distribution de probabilité considérée sur les arbres binaires de recherche de taille n est comme suit. Soit un arbre binaire de recherche A de taille n dont les deux sous-arbres sont notés D et G, la probabilité de A est définie de manière récursive par

$$\Pr(A) = \frac{1}{n} \Pr(G) \Pr(D).$$

La figure 2.2 montre un exemple d'arbre binaire de recherche de taille 19.

2.3.2 Valeurs de divers paramètres

Cette section rappelle le nombre d'arbres binaires de taille n, les valeurs moyennes de la longueur de cheminement interne de la hauteur et du coût d'insertion et enfin décrit le profil moyen d'un arbre binaire de recherche.

Nombre

Lorsque les arbres binaires de recherche sont étiquetés de manière canonique, ils sont uniquement déterminés par leur forme d'arbre. On dénombre ici le nombre de *forme* d'arbres binaires de recherche

de taille n, qui est en fait le nombre d'arbres binaires.

Le nombre d'arbres binaires de taille n est un paramètre classique, et est rappelé et prouvé ici.

Proposition 2.5 *Le nombre d'arbres binaires de taille n est égal au nombre de Catalan C_n :*

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Preuve. Notons pour l'instant B_n le nombre d'arbres binaires de taille n. La série génératrice de B_n, définie par $B(z) = \sum B_n z^n$ satisfait l'équation fonctionnelle

$$B(z) = 1 + zB(z)^2,$$

car un arbre binaire est soit vide (codé par 1) soit composé d'une racine (codée par z) et de deux sous-arbres (codés par $B(z)^2$). Cette équation de degré 2 possède deux solutions. La condition initiale $B_0 = 1$ permet de sélectionner celle qui convient, et on obtient alors $B(z) = \frac{1-\sqrt{1-4z}}{2z}$.

Cette série génératrice est la série génératrice des nombres de Catalan, ce qui prouve la proposition. On peut observer de plus, grâce à la formule de Stirling, ou l'analyse de singularité qu'asymptotiquement $C_n \sim \frac{4^n}{\sqrt{\pi} n^{3/2}}$.

Les nombres de Catalan interviennent régulièrement dans des problèmes de combinatoire, et sont largement étudiés, voir [AS73, Com74, GKP94]. ■

LCI - Insertion - Profil

L'analyse de la longueur de cheminement interne (noté *LCI*), du coût d'insertion et du profil est brièvement résumée ici. Une étude détaillée de ces paramètres peut être trouvée dans le livre de Mahmoud [Mah92].

On dit qu'un sommet est à profondeur k si la distance (en nombre d'arêtes) qui le sépare de la racine est égale à k. Par convention, la racine est à profondeur 0.

La *longueur de cheminement interne* (lci) est la somme des profondeurs de tous les sommets de l'arbre.

Proposition 2.6 *(i) L'espérance de la longueur de cheminement interne d'un arbre binaire de recherche de taille n notée L_n vérifie*

$$L_n = 2n \log n + O(n).$$

(ii) Le coût moyen d'une insertion dans un ABR de taille n noté I_n est

$$I_n = 2 \log n + O(1).$$

(iii) Soit $m_{k,n}$ le nombre moyen de sommets situés à profondeur k. Le polynôme des niveaux moyen est défini par $P_n(u) = \sum_k m_{k,n} u^k$. La variable aléatoire Y_n qui code le profil des arbres binaires de recherche dont la loi est définie comme suit. La probabilité que Y_n soit égale à k est $m_{k,n}/P_n(1)$. La variable aléatoire Y_n est asymptotiquement gaussienne de moyenne $2 \log n$ et de variance $2 \log n$.

La suite est consacrée à la preuve de cette proposition.

Polynôme des niveaux. Le polynôme des niveaux moyen est égal à $P_n(u) = \sum_k m_{k,n} u^k$, où $m_{k,n}$ est défini dans l'énoncé de la proposition comme étant la proportion moyenne de sommets à profondeur k. Ce polynôme code le nombre moyen de sommets à profondeur k dans un ABR. L'ensemble des coefficients $m_{k,n}$ permet d'exprimer le "profil" de l'arbre moyen. Certains paramètres s'expriment en fonction du polynôme des niveaux. En particulier, l'espérance de la longueur de cheminement interne est égale à $P_n'(1)$, et l'analyse de $P_n(u)$ au voisinage de $u = 1$ permet d'obtenir le point (iii) de la proposition.

Le polynôme $P_n(u)$ satisfait la récurrence

$$P_n(u) = 1 + \frac{1}{n} u \sum_{i=1}^{n} \left(P_{i-1}(u) + P_{n-i}(u) \right),$$

car il y a un sommet au niveau 0, et ensuite si la racine est le i-ième élément (ce qui a une probabilité $1/n$) l'arbre possède deux sous-arbres de tailles respectives $i-1$ et $n-i$. La profondeur d'un sommet dans un sous-arbre est sa profondeur dans l'arbre moins 1, donc on multiplie la contribution du fils P_{i-1} ou P_{n-i} par u pour traduire ce décalage.

On définit ensuite la série génératrice double $P(z,u) := \sum_n P_n(u) z^n$. La récurrence satisfaite par $P_n(u)$ se traduit en une équation différentielle linéaire pour $P(z,u)$ qui est

$$\frac{\partial}{\partial z} P(z,u) - \frac{2u}{1-z} P(z,u) = \frac{1}{(1-z)^2}.$$

Cette équation différentielle se résout sans difficulté grâce à la condition initiale $P(0,u) = 0$, qui exprime simplement le fait qu'un arbre de taille 0 n'a aucun sommet, et on obtient

$$P(z,u) = \frac{1}{1-2u} \left(\frac{1}{1-z} - \frac{1}{(1-z)^{2u}} \right). \tag{2.7}$$

Longueur de cheminement. La longueur de cheminement interne moyenne L_n est égale à $P_n'(1)$, car $L_n = \sum k m_{k,n}$. Cette valeur s'obtient en dérivant la série génératrice $P(z,u)$, puis en évaluant le résultat en $u = 1$, et enfin en extrayant le coefficient de $[z^n]$. Le résultat est

$$\begin{aligned} L_n &= [z^n] \frac{2}{(1-z)^2} \log\left(\frac{1}{1-z}\right) - \frac{2}{(1-z)^2} + \frac{2}{1-z} \\ &= 2n \log n + 2(\gamma - 2)n + 2 \log n + 2\gamma + 1 + o(1), \end{aligned}$$

où γ est la constante d'Euler. Le point (i) de la proposition est alors prouvé.

La longueur de cheminement externe d'un arbre est la somme des profondeurs des feuilles. On note LE_n la longueur de cheminement externe moyenne d'un arbre binaire de recherche. Les longueurs de cheminement externe et interne (lce et lci) d'un arbre de taille n sont reliées par la relation $lce = lci + 2n$, qui se vérifie facilement par induction. Cela entraîne $LE_n = 2n \log n + O(n)$.

Recherche - Insertion. Le coût de recherche d'un élément est défini comme le nombre de comparaisons effectuées lors de la recherche. Pour rechercher un élément dans un arbre binaire de recherche, l'algorithme donné par la plupart des références suit les grandes lignes suivantes.

-Si l'arbre est vide, renvoyer un résultat d'échec, sinon :
  -Tester si l'élément est strictement inférieur à la racine ;
    -Si oui, rechercher dans le sous-arbre gauche ;

-Si non, tester si l'élément est égal à la racine ; si ce n'est pas le cas, le rechercher dans le sous arbre droit.

Le coût de cette procédure de recherche est en moyenne de $2\log n$ comparaisons *ternaires* (qui permettent de distinguer $<$, $=$ et $>$), soit $3\log n$ comparaisons *binaires*. Ce point n'est que rarement explicité dans la littérature.

Arne Andersson dans [And91] propose un algorithme de recherche qui coûte en moyenne $2\log n$ comparaisons *binaires*. Cet algorithme se contente de décider si l'élément recherché est strictement inférieur ($<$) ou supérieur ($\geq$) à la racine, et poursuit ensuite selon le cas sa recherche dans le fils droit ou gauche, en se souvenant dans le cas $\geq$ que la racine est candidate pour être égale à l'élément recherché. La force de cet algorithme est qu'un candidat chassant l'autre, à la fin il n'y a besoin que d'une comparaison supplémentaire avec l'unique candidat en lice. L'algorithme en pseudo code est :

-*candidat* :=`null`

-Tester si l'élément est strictement inférieur à la racine ;

-Si oui, rechercher dans le sous-arbre gauche ;

-Si non, *candidat* $\leftarrow$ *racine*, et rechercher dans le sous arbre droit.

-Si *candidat* $\neq$ `null`, tester si l'élément est égal au *candidat*.

Le coût de la recherche d'un élément est alors équivalent à la profondeur moyenne d'une feuille, laquelle est $2\log n + O(1)$.

Le coût d'une insertion est défini comme le nombre de comparaisons effectuées. L'insertion dans un ABR se fait au niveau des feuilles. L'algorithme consiste à d'abord rechercher l'élément, puis à l'insérer dans la feuille où la recherche s'est terminée. Le coût moyen d'une insertion est donc équivalent au coût moyen d'une recherche, soit $I_n = 2\log n + O(1)$, ce qui prouve (*ii*).

Profil. Pour étudier le profil de l'arbre, on revient à la série génératrice du polynôme des niveaux $P(z,u)$ explicitée dans l'équation (2.7). Grâce à des méthodes d'analyse de singularités, on peut extraire le coefficient de $[z^n]$, et obtenir que lorsque le paramètre u est dans un voisinage de 1, on a uniformément

$$P_n(u) \sim \frac{1}{1-2u}\left(1 - \frac{n^{2u-1}}{\Gamma(2u)}\right) \sim \frac{n^{2u-1}}{2u-1}$$

En appliquant le théorème des quasi-puissances de Hwang [Hwa94], on trouve que la loi de la variable aléatoire Y_n (définie par $\Pr(Y_n = k) = m_{k,n}/P_n(1)$) est asymptotiquement gaussienne de moyenne $2\log n$ et de variance $2\log n$, lorsque n tend vers l'infini. Ce qui prouve le point (*iii*) de la proposition.

Hauteur

Proposition 2.7 *L'espérance de la hauteur d'un arbre binaire de recherche de taille n est équivalente à $c\log n$, où la constante c est définie par $\left(\frac{2e}{c}\right)^c = e$ et vérifie $c = 4.311\ldots$.*

La preuve de cette proposition est assez délicate, car la hauteur n'est pas un paramètre additif, contrairement à tous les paramètres étudiés jusqu'ici. Luc Devroye a établi ce résultat en 1987 [Dev87] par des méthodes de processus de branchements, et Michael Drmota l'a prouvé en 2001 [Drm01] par des méthodes analytiques, en obtenant de plus des bornes sur le deuxième ordre asymptotique. Finalement Bruce Reed a obtenu récemment (en 2003) [Ree03] le deuxième ordre asymptotique. Il montre que la hauteur moyenne d'un ABR de taille n est $c\log n - d\log\log n + O(1)$, où $d = 1.953\ldots$, lorsque n tend vers l'infini.

Ce résultat décrit l'espérance du temps d'accès maximal à un élément dans un arbre binaire, ou encore la profondeur de récursion maximale moyenne de l'algorithme quicksort.

2.3.3 Profondeur moyenne du k-ième élément

Dans cette sous-section, on se propose de calculer la profondeur moyenne du k-ième sommet dans un ABR de taille n, qu'on note $g_{n,k}$. On rappelle que les sommets sont supposés numérotés canoniquement de 1 jusqu'à n. De façon naturelle, ce qu'on nomme le k-ième élément est alors celui qui porte le numéro k. Grâce à la décomposition récursive d'un ABR, on peut écrire la récurrence

$$g_{n,k} = \frac{1}{n}\sum_{i=1}^{k-1}(1+g_{n-i,k-i}) + \frac{1}{n}\sum_{i=k+1}^{n}(1+g_{i-1,k})\,.$$

Cette récurrence se lit de la manière suivante. Le sommet numéro i est la racine avec probabilité $1/n$. Si k est strictement plus petit que i, alors on recherche le k-ième sommet dans un arbre de taille $i-1$, en prenant en compte le fait qu'on a déjà suivi une branche. Si k est strictement plus grand que i, alors on est ramené à chercher dans un arbre de taille $n-i$ le sommet numéro $k-i$. Finalement si $k=i$, c'est terminé, on est à profondeur 0.

Cette récurrence bivariée peut être transformée en une équation différentielle portant sur la série génératrice $G(z,u)=\sum g_{n,k}z^n u^k$:

$$\frac{\partial G(z,u)}{\partial z} = G(z,u)\left(\frac{u}{1-zu}+\frac{1}{1-z}\right)+\left(\frac{1}{(1-z)^2}-\frac{u}{(1-zu)^2}\right)\frac{u}{1-u}.$$

Comme cette équation n'est différentielle que par rapport à la variable z, on peut la résoudre grâce à la méthode de variation de la constante, avec comme condition initiale $G(0,u)=0$. On obtient la série génératrice

$$G(z,u) = \frac{1}{1-z}\frac{1}{1-zu}\left(u\ln\frac{1}{1-z}+\ln\frac{1}{1-zu}-zu\right)$$

On peut alors extraire de façon explicite le coefficient de $[z^n u^k]$ de cette série génératrice, ce qui nous permet d'énoncer la proposition suivante

Proposition 2.8 *La profondeur moyenne du k-ième sommet dans un arbre binaire de recherche de taille n est*

$$g_{n,k} = H_k + H_{n-k},$$

où H_k désigne le k-ième nombre harmonique, $H_k=\sum_{i=1}^{k}\frac{1}{i}$.

Cette valeur est équivalente asymptotiquement à $\log k+\log(n-k)$, ce qui permet de constater que si k vérifie $\alpha n<k<(1-\alpha)n$, pour une constante $0<\alpha<0.5$, alors $g_{n,k}\sim 2\log n$.

2.3.4 Un résumé en images

La figure 2.3 résume une partie de cette section. La première courbe, à gauche, montre le tracé de la profondeur moyenne du k-ième élément. On observe bien le plateau autour de la profondeur $2\log n$. On voit également que les éléments dont les indices sont les plus petits ou les plus grands se trouvent en moyenne situés à moindre profondeur dans l'arbre que les autres.

Le deuxième tracé de la figure 2.3 montre la quantité de sommets situés en moyenne à chaque niveau, c'est-à-dire le profil moyen de l'arbre. On observe qu'un arbre binaire de recherche moyen est ventru, avec la plupart de ses sommets situés à une profondeur proche de $2\log n$.

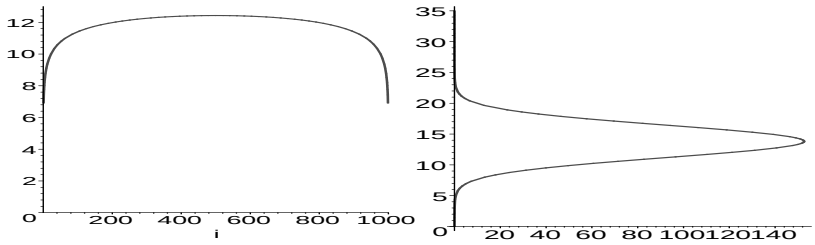

FIG. 2.3 – Un arbre binaire de recherche moyen de taille $n = 1000$, vu de face et de profil (en projection)

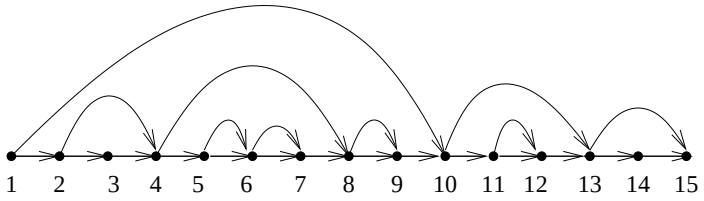

FIG. 2.4 – Une liste à sauts de taille 15

2.4 Listes à sauts

Les *listes à sauts*, ou *jumplists*, sont une structure de donnée inspirée des skiplists, dues à Pugh [Pug90], d'où le nom de jumplists. Cette structure a été inventée par Frédéric Cazals et Hervé Brönnimann, et ma contribution est d'avoir mené à bien l'analyse des paramètres usuels de cette structure. L'article [BCD03] propose un exposé complet concernant les listes à sauts.

2.4.1 Définition - Présentation

Définition

Une liste à sauts est une structure de donnée qui fournit les opérations usuelles d'un dictionnaire c'est-à-dire insertion, suppression et recherche. C'est une liste chaînée ordonnée, dont les nœuds possèdent un pointeur nommé `next` et un pointeur supplémentaire nommé `jump`, qui permet d'avancer plus rapidement dans la liste. On suppose que le premier élément de la liste a un rôle de sentinelle, c'est-à-dire que tous les éléments insérés ont une valeur plus grande que celle du premier nœud. Une liste à sauts de taille 15 est représentée sur la Figure 2.4.

Les règles que doivent respecter les pointeurs `jump` sont les suivantes :

-Chaque nœud est point de départ d'un pointeur `jump`, sauf le dernier

-Un nœud est point d'arrivée d'au plus un pointeur `jump`

-Deux pointeurs ne peuvent se croiser

-Un pointeur `jump` peut être égal à `null`, si c'est la seule solution compatible avec les règles précédentes.

Ces règles font que les pointeurs `jump` forment un système d'arches au dessus de la liste chaînée. Pour parcourir une liste à sauts, on peut alors décider à chaque nœud si on suit le pointeur de la liste chaînée (on marche), ou si on utilise le pointeur `jump` (on saute). Si les arches sont bien disposées, cela permet d'accéder à un élément de façon très rapide.

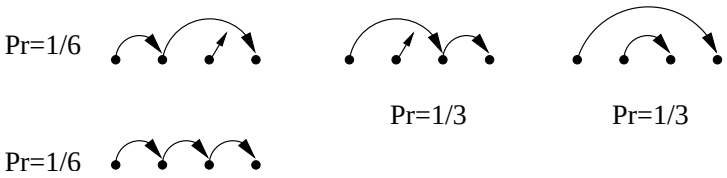

FIG. 2.5 – Probabilités des listes à sauts de taille 4.

Liste à sauts randomisée

Les listes à sauts permettent d'accélérer les requêtes usuelles par rapport à une liste chaînée, si les pointeurs `jump` sont bien disposés. Cette "bonne" disposition doit pouvoir être conservée de façon simple et rapide lors de l'insertion ou la suppression d'un élément. L'idée, inspirée des skiplists randomisées, est de manipuler des listes à sauts randomisées. On dit qu'une liste à sauts est randomisée si le premier pointeur `jump` atterrit avec probabilité uniforme sur tous les éléments qui le suivent, et que de plus la liste située sous cette arche et la liste située après cette arche sont également des listes à sauts randomisées.

Les règles de construction d'une liste à sauts *randomisée* sont les suivantes.

1. On construit tout d'abord la liste chaînée des éléments ordonnés notés a_k, k allant de 1 à n.
2. Pour le nœud a_k, en procédant par ordre croissant k allant de 1 à la taille de la liste à sauts, on crée un pointeur `jump` qui part de a_k, et qui atterrit avec probabilité uniforme parmi tous les nœuds qu'il peut atteindre sans croiser d'autres pointeurs `jump` précédemment crées. (Si deux pointeurs ont le même point d'atterrissage, on considère qu'il y a croisement.) Si l'ensemble des possibilités est vide alors le pointeur se voit affecter la valeur `null`.

Pour rechercher un élément dans une liste à sauts, on regarde comment il se situe par rapport à l'arrivée du premier pointeur `jump`. Soit il est supérieur et on utilise ce pointeur `jump`, soit il est égal et c'est gagné, soit on marche, c'est-à-dire qu'on utilise le pointeur `next`. Les algorithmes d'insertion et de suppression qui permettent de maintenir les propriétés d'aléa utilisent des idées assez fines. L'algorithme d'insertion est présenté dans la section 2.4.3 ; l'algorithme de suppression est l'inverse du processus d'insertion et n'est pas décrit ici. L'article [BCD03] donne le détail de ces deux algorithmes.

Cette structure de donnée présente de plus l'avantage de permettre de rechercher tous les éléments d'un intervalle très rapidement en suivant simplement les pointeurs `next`.

Modèle probabiliste

On suppose maintenant que les nœuds d'une liste à sauts sont étiquetés canoniquement par les entiers de 1 à n. Une liste à sauts est alors uniquement définie par sa forme.

Le modèle probabiliste choisi pour analyser les listes à sauts est le suivant. On considère une liste à sauts C de taille n dont la liste issue du pointeur `next` est notée N, et la liste issue du pointeur `jump` est notée J. La probabilité de C s'exprime en fonction de la probabilité de ses sous-listes, soit

$$\Pr(C) = \frac{1}{n-1} \Pr(N) \Pr(J).$$

Cette propriété est l'équivalent de la propriété de décomposabilité des ABR, qui permet d'appliquer le principe "diviser pour régner".

La figure 2.5 donne cette loi de probabilité sur toutes les listes à sauts de taille 4.

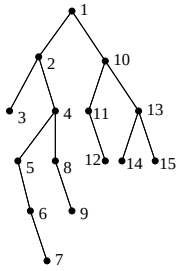

FIG. 2.6 – L'arbre binaire correspondant à la liste à sauts de la figure 2.4

Correspondance liste à sauts - arbre binaire.

Une liste à sauts peut être vue comme un arbre binaire, ayant comme racine le premier élément de la liste, comme fils gauche l'arbre binaire construit récursivement à partir de la sous liste allant du 2^e élément inclus jusqu'à l'arrivée du pointeur `jump` exclu, et comme fils droit, l'arbre binaire construit à partir de la sous liste commençant au pointeur `jump`. Cet arbre binaire permet de ranger dans le fils gauche tous les éléments qu'on atteint via le pointeur `next` et dans le fils droit tous ceux auxquels on accède par le pointeur `jump`. La figure 2.6 montre l'arbre binaire qui correspond à la liste à sauts de la figure 2.4.

Les performances des algorithmes sur les listes à sauts sont reliées à la forme de cet arbre. Le nombre de pointeurs qu'il faut suivre pour atteindre un élément est par définition sa profondeur dans l'arbre binaire associé, et par conséquent le temps de recherche moyen est égal à la profondeur moyenne des éléments dans l'arbre. On a donc tout intérêt à ce que l'arbre soit bien équilibré pour minimiser le temps moyen de recherche.

On peut associer un arbre binaire à une liste à sauts, mais cette correspondance n'est pas surjective, car l'arbre binaire obtenu présente la particularité suivante. Si un nœud n'a qu'un seul fils, alors, avec les conventions choisies, c'est nécessairement le fils gauche qui est vide.

Les sections suivantes étudient des paramètres des listes à sauts via le point de vue des arbres binaires. On commence par compter le nombre de formes de listes à sauts de taille fixée dans la section 2.4.2, puis la section 2.4.3 analyse le coût moyen de l'algorithme d'insertion. La section 2.4.4 s'intéresse à la longueur de cheminement interne moyenne et au profil moyen de l'arbre binaire associé à une liste à sauts, et la section 2.4.5 fournit une majoration pour la hauteur moyenne d'une liste à sauts. La dernière partie consacrée aux listes à sauts (2.4.6) calcule la profondeur moyenne du k-ième élément.

2.4.2 Nombre de listes à sauts

On appelle *forme* d'une liste à sauts le graphe obtenu lorsqu'on efface les étiquettes d'une liste à sauts.

La figure 2.7 présente toutes les formes de listes à sauts de taille inférieure à 5. On constate que le nombre de formes de listes à sauts, groupées selon la taille est $1, 1, 2, 4, 9$. La proposition suivante précise cette quantité lorsque la taille est n.

Proposition 2.9 *Soit j_n le nombre de formes de listes à sauts de taille n étiquetées de manière canonique par les entiers entre 1 et n, et $j(z)$ la série génératrice $\sum j_n z^n$. On a*

$$j(z) = \frac{1 + z - \sqrt{1 - 2z - 3z^2}}{2z},$$

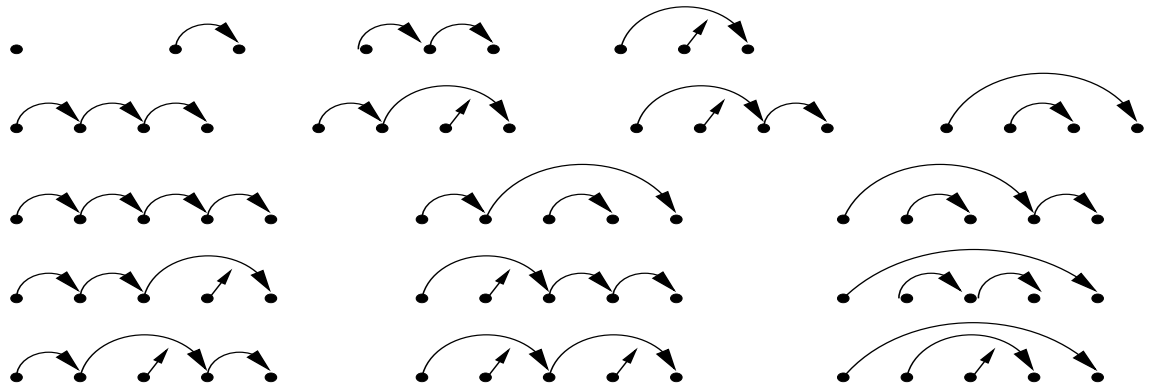

FIG. 2.7 – Jumplists de taille 1 à 5. (Les pointeurs `next` sont omis)

ainsi que les résultats sur les coefficients :

$$j_n = M_{n-1} \qquad et \qquad j_n \sim 3^n \frac{\sqrt{3}}{\sqrt{\pi}} \frac{1}{n^{3/2}},$$

où M_n désigne le n-ième nombre de Motzkin.

Ces résultats peuvent être comparés à ceux des arbres binaires, présentés dans la section précédente. Le nombre d'arbres binaires de taille n est donné par les nombres de Catalan C_n qui vérifient

$$C_n = \frac{1}{n+1}\binom{2n}{n} \qquad \text{et} \qquad C_n \sim \frac{4^n}{\sqrt{\pi} n^{3/2}}.$$

La série génératrice de ces nombres notée $C(z)$ est égale à $\frac{1-\sqrt{1-4z}}{2z}$. Ces résultats ont des formes similaires, et on remarque surtout qu'il y a asymptotiquement beaucoup moins de formes de listes à sauts que d'arbres binaires, car le terme exponentiel passe de 3^n à 4^n.

Preuve. La preuve de la proposition se fonde sur la décomposition combinatoire des listes à sauts. La notation $\mathcal{J}$ désigne l'ensemble des formes des listes à sauts, toutes tailles confondues, y compris de la liste à sauts vide, qui est notée ϵ. La liste à sauts de taille 1 est représentée par $\bullet$.

$$\mathcal{J} = \epsilon + \bullet + \bullet \times \mathcal{J} \times (\mathcal{J} - \epsilon).$$

Cette équation exprime qu'une liste à sauts est soit vide, soit de taille 1, soit composée d'une racine, d'une liste à sauts `next` éventuellement vide, et d'une liste à sauts jump qui ne peut être vide. Cette décomposition ensembliste se traduit d'après les méthodes symboliques [FS05] par une équation sur les séries génératrices

$$j(z) = 1 + z + zj(z)(j(z) - 1).$$

Cette équation de degré 2 admet deux solutions dont une seule $j(z) = \frac{1+z-\sqrt{1-2z-3z^2}}{2z}$ est analytique en 0. Ceci prouve la première partie de la proposition.

On reconnaît alors que la série génératrice est à un facteur z près égale à la série génératrice des nombres de Motzkin, pour lesquels on dispose de nombreux résultats, voir par exemple Stanley [Sta98]. Les nombres de Motzkin dénombrent les chemins de Motzkin de taille n. Ces chemins sont définis dans $\mathbb{N}^2$ par un point de départ qui est $(0,0)$, un point d'arrivée $(n,0)$, des déplacements possibles qui sont au nombre de trois : un pas nord-est $(1,1)$, un pas est $(1,0)$ et un pas sud-est

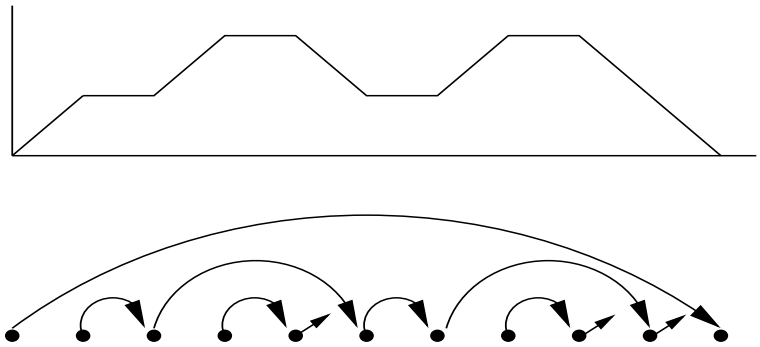

FIG. 2.8 – Un exemple de la bijection listes à sauts - Chemins de Motzkin.

$(1, -1)$, avec la contrainte supplémentaire que ce chemin ne doit jamais descendre sous l'axe $y = 0$, c'est-à-dire que chaque niveau descendu doit avoir été monté auparavant.

La comparaison avec les résultats asymptotiques classiques (ou encore l'analyse de singularité directe de $j(z)$) donne la forme asymptotique de la seconde partie de l'énoncé.

■

L'égalité $j_n = M_{n-1}$ exprime qu'il y a autant de chemin de Motzkin de taille $n-1$ que de listes à sauts de taille n. Ce résultat peut être prouvé directement de façon bijective comme indiqué ci-dessous. La figure 2.8 représente une liste à sauts et le chemin de Motzkin correspondant par cette bijection.

Bijection. Partant d'une liste à sauts de taille n, on trace une ligne verticale passant par chacun de ses nœuds. Le niveau du chemin de Motzkin au point i est le nombre de pointeurs `jump` que traverse la ligne verticale passant par le nœud i. Le chemin obtenu est bien de longueur $n-1$, mais il faut encore montrer que c'est bien un chemin de Motzkin. Il y a trois cas de figures différents possibles, soit le nœud est point de départ d'un pointeur `jump` trivial, auquel cas le niveau du chemin descend de 1 (pas sud-est), soit le nœud est point de départ d'un pointeur `jump` non-trivial, et il y a alors deux possibilités : le nœud est l'arrivée d'un pointeur `jump`, auquel cas le niveau est stable (pas est), ou il ne l'est pas auquel cas le niveau monte (pas nord-est). Il y a toujours un nombre positif de pointeurs au dessus de chaque nœud, donc le chemin est toujours au dessus de l'axe $y = 0$. Le point de départ du chemin ainsi construit est $(0, 0)$ par définition et son point d'arrivée est à l'altitude 0, car il n'y a aucun pointeur `jump` au dessus du dernier élément. C'est donc bien un chemin de Motzkin.
La construction réciproque se base sur les mêmes principes.

2.4.3 Insertion

La difficulté de l'insertion provient du fait qu'on doit maintenir les propriétés d'aléa de la liste à sauts. Par exemple, pour une liste à sauts de taille n, chaque nœud, excepté le premier a une probabilité $1/(n-1)$ d'être le point d'arrivée du pointeur `jump` issu du premier élément. Avec l'insertion d'un nouvel élément, chaque élément, y compris le nouveau, est l'arrivée de ce pointeur avec probabilité $1/n$. Il faut donc a priori reconstruire les arches de la liste à sauts pour insérer un élément. L'algorithme d'insertion va cependant le plus souvent possible tenter de recycler les arches préexistantes.

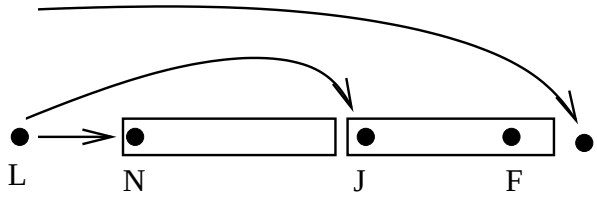

FIG. 2.9 – Insertion : notations.

Algorithmique

On présente un premier algorithme d'insertion qui préserve les propriétés d'aléa, puis une variante qui l'optimise.

Premier algorithme. Notons x l'élément à insérer, lequel est inséré dans une liste nommée L, dont les sous listes `next` et `jump` sont nommées respectivement N et J ; cf figure 2.9. On note n la taille de la liste à sauts L avant l'insertion. On confond systématiquement un élément et la sous-liste qui part de cet élément.

L'algorithme procède comme suit :

- Le nouvel élément est point d'arrivée du pointeur `jump` partant de L avec une probabilité $1/n$. Dans ce cas de figure, on doit alors reconstruire la liste $L \cup \{x\}$ pour restaurer les propriétés de randomisation de la liste à sauts. Cela est fait en utilisant l'algorithme de construction déjà présenté.
- Le deuxième cas est qu'avec probabilité $1 - 1/n$, le pointeur `jump` n'est pas modifié.
  - Si x doit être inséré juste après L, et avant N (donc en deuxième position), alors on reconstruit toute la liste $N \cup \{x\}$, la liste J n'étant alors pas affectée par ce changement.
  - Si, au contraire la position de x est après N, alors on insère récursivement dans la sous liste N ou J selon la position de x.

Cet algorithme maintient bien les propriétés de randomisation des listes à sauts.

Cependant les cas de figures où l'on doit reconstruire la liste à sauts sont trop fréquents et trop coûteux. Par exemple, le coût de l'insertion d'un élément en deuxième position, juste après la racine (L) est équivalent à n, voir [BCD03] pour les détails.

Algorithme optimisé. Il est possible d'améliorer l'algorithme précédent, afin de diminuer le coût moyen de l'insertion. L'algorithme optimisé proposé ici procède comme le premier algorithme, sauf dans un cas particulier.

Le cas de figure amélioré ici et celui où l'insertion a lieu juste après la racine. L'intuition est qu'au lieu de tout reconstruire à partir de rien, on va pouvoir recycler certaines arches. On suppose être dans la situation où le pointeur `jump` issu de la racine n'est pas modifié (ce qui a une probabilité $1 - 1/n$). Les notations sont comme il est précisé sur la figure 2.10. Le schéma (a) présente la situation avant insertion, et le schéma (b) la situation après insertion de la valeur, mais avant la mise à jour des arches. L'élément x s'insère juste après la racine, et le pointeur `jump` partant de C est inchangé.

L'algorithme optimisé crée alors le pointeur `jump` qui part de x. On note k la taille de la liste N. Il y a deux cas de figures.

- Avec probabilité $1/k$, le pointeur `jump` de x atterrit sur N, et les arches suivantes peuvent être préservées (cas (c)).

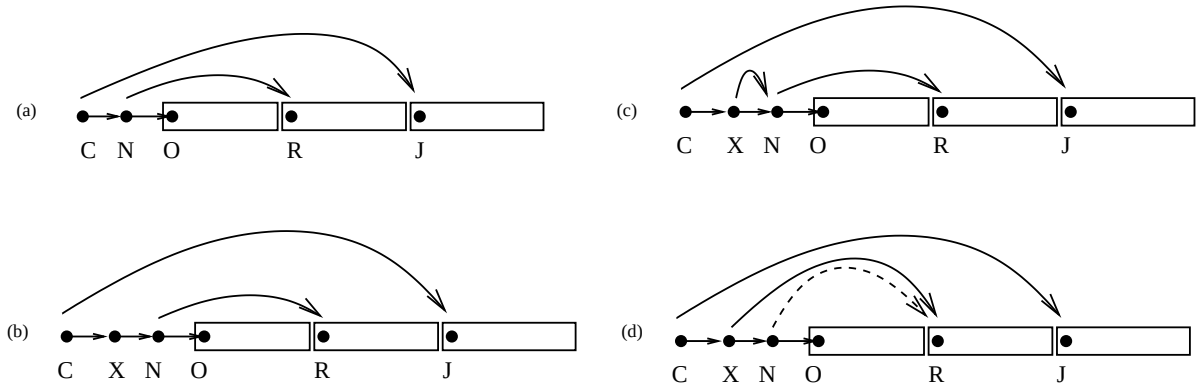

FIG. 2.10 – Usurpation d'arches

– Avec probabilité $1 - 1/k$ l'arrivée du pointeur `jump` est après N. Alors x usurpe l'arche `jump` qui part de N, car cette arche possède la distribution adéquate (cas d). La liste `next` de x doit alors être réorganisée, ce qui est fait de façon récursive.

L'algorithme optimisé préserve les propriétés de distribution des arches dans la liste à sauts.

Coût moyen

Proposition 2.10 *L'espérance du coût d'une insertion dans une liste à sauts de taille n est asymptotiquement équivalente à $2 \log n$.*

Le reste de ce paragraphe est consacré à la preuve de cette proposition. On énonce d'abord un lemme préliminaire qui correspond au cas où l'insertion a lieu juste après le premier élément de la liste, et où on est en situation d'utiliser la stratégie d'usurpation d'arches.

Lemme 2.2 *Le coût moyen d'une insertion en deuxième position dans une liste à sauts aléatoire par l'algorithme optimisé est asymptotiquement équivalent à $\log n$, où n est la taille de la liste next..*

Notons S_n le nombre moyen de pointeurs mis à jour lors d'une insertion faite juste après le premier nœud, et donc basé sur l'algorithme d'usurpation d'arches. Comme l'algorithme d'insertion est récursif, on peut écrire une équation de récurrence sur le coût S_n, qui se décompose selon les positions du pointeur `jump` de x

$$S_n = \frac{1}{n}1 + \left(1 - \frac{1}{n}\right)\frac{1}{n-1}\sum_{i=2}^{n}(1 + S_{i-2}).$$

Cette récurrence code le fait qu'avec une probabilité $1 - 1/n$ on usurpe l'arche et on continue récursivement et qu'avec une probabilité $1/n$, on se limite à créer une arche de longueur 1.

Cette récurrence peut être étudiée grâce aux méthodes présentées dans la section 1.1.2. La série génératrice $S(z)$ satisfait une équation différentielle, écrite ici avec sa solution.

$$S'(z) - \frac{z}{1-z}S(z) = \frac{1}{(1-z)^2}, \quad S(z) = \frac{(\mathrm{Ei}(1-z) - \mathrm{Ei}(1))e^{1-z}}{1-z}, \tag{2.8}$$

où $\mathrm{Ei}(x)$ désigne l'exponentielle intégrale, définie par

$$\mathrm{Ei}(x) = \int_1^\infty \frac{e^{-xt}}{t}dt, \tag{2.9}$$

voir [AS73]. Cette série génératrice vérifie $S(z) \sim_{z\to 1} \frac{1}{1-z}\log\frac{1}{1-z}$, ce qui permet d'obtenir le comportement asymptotique de S_n par analyse de singularité :

$$S_n \sim \log n. \tag{2.10}$$

Pour prouver la proposition 2.10, on écrit une relation de récurrence sur T_n représentant l'espérance du coût total d'insertion d'un élément dans toutes les positions possibles

$$T_n = n\frac{1}{n}n + \left(1-\frac{1}{n}\right)\frac{1}{n-1}\sum_{i=2}^{n}\left(S_{i-2}+T_{n-i+1}+T_{i-2}\right).$$

Cette récurrence traduit le comportement de l'algorithme, avec la variable i qui dénote le point d'arrivée du premier pointeur `jump`. On reprend les notations de la figure 2.9 pour décrire cette décomposition. Avec probabilité $1/n$ on doit reconstruire la liste entière, ce qui a un coût n, et ce pour tous les éléments. Ensuite la liste n'est pas entièrement reconstruite avec probabilité $\left(1-\frac{1}{n}\right)$, et i est point d'arrivée du pointeur `jump` avec probabilité $1/(n-1)$. Les différentes positions possibles pour l'insertion sont :

- en deuxième position, ce qui a un coût moyen égal à S_{i-2}, par le lemme 2.2 ;
- dans la liste à sauts N accessible par le pointeur `next`, ce qui a un coût moyen égal à T_{i-2} car on insère dans une liste à sauts de taille $i-2$;
- dans la liste à sauts J accessible par le pointeur `jump`, ce qui a un coût moyen égal à T_{n-i+1} car on insère dans une liste à sauts de taille $n-i+1$.

On récrit alors la récurrence de façon plus lisible

$$T_n = n + \frac{1}{n}\sum_{i=2}^{n}\left(S_{i-2}+T_{n-i+1}+T_{i-2}\right)$$

La condition initiale de cette récurrence est $T_0 = 0$, car on ne manipule aucun pointeur pour insérer dans une liste vide.

Etant donné qu'on connaît la série génératrice associée à S_n, on peut transformer l'équation de récurrence de T_n en équation différentielle vérifiée par $T(z) = \sum T_n z^n$, et on obtient

$$T'(z) - T(z)\frac{1+z}{1-z} = \frac{2z}{(1-z)^3} + \frac{1}{(1-z)^2} + \frac{zS(z)}{1-z}.$$

Cette équation se résout à nouveau par variation de la constante en utilisant l'équation (2.8), et la solution à cette équation différentielle est

$$T(z) = \frac{e^{1-z}(1+z)(\mathrm{Ei}(1-z)-\mathrm{Ei}(1))}{(1-z)^2}$$

et vérifie $T(z) \sim_{z\to 1} \frac{2}{(1-z)^2}\log\left(\frac{1}{1-z}\right)$, d'où on extrait la formule asymptotique $T_n \sim 2n\log n$. Ceci achève donc la preuve de la proposition car le coût moyen d'insertion, qui vaut T_n/n, est donc équivalent à $2\log n$.

2.4.4 LCI - Loi limite du profil

Les paramètres déjà définis pour les arbres sont étendus aux listes à sauts, grâce à la représentation des listes à sauts sous forme d'arbre binaire. La longueur de cheminement interne (lci) est un paramètre intéressant pour les listes à sauts, car il permet d'accéder au coût moyen d'une recherche.

Pour étudier la lci et le profil des listes à sauts, on introduit le polynôme des niveaux qui permet de coder la forme de l'arbre associé.

Définition 2.2 *Le polynôme des niveaux d'un arbre est* $\sum_{v \text{ noeud}} u^{profondeur(v)}$.

Le coefficient de u^k dans ce polynôme est le nombre de nœud au niveau k dans l'arbre. Le polynôme des niveaux moyen des listes à sauts de taille n est alors défini comme suit.

Définition 2.3 *Soit* $s_{k,n}$ *le nombre moyen de nœuds à profondeur* k *parmi toutes les listes à sauts de taille* n. *Le polynôme des niveaux moyen des listes à sauts est alors défini par* $S_n(u) = \sum_{k\geq 0} s_{k,n}u^k$. *La série génératrice associée est* $S(z,u) = \sum_{n\geq 0} S_n(u)z^n$.

La longueur de cheminement interne moyenne des listes à sauts est égale à $S_n'(1)$ et peut aisément être obtenue à partir de $S(z,u)$. Cette série génératrice est également utilisée pour étudier le profil des listes à sauts.

Les méthodes utilisées dans cette section sont parfois très proches de celles utilisées pour l'étude des arbres binaires de recherche, et on ne les rappelle alors que brièvement.

Longueur de cheminement interne. Le pointeur `jump` du premier élément atterrit avec probabilité uniforme sur tous les éléments de la liste, excepté lui même. Les tailles des sous-arbres droit et gauche de l'arbre binaire associé sont donc k et $n-k-1$ $(1 \leq k \leq n-1)$ avec probabilité uniforme $\frac{1}{n-1}$. La suite $S_n(u)$ satisfait donc la récurrence

$$S_n(u) = 1 + \frac{u}{n-1}\left(\sum_{k=1}^{n-1} S_k(u) + S_{n-k-1}(u)\right). \tag{2.11}$$

La série génératrice $S(z,u)$ satisfait l'équation différentielle

$$S'(z,u) - S(z,u)\left(\frac{1}{z} + u\frac{1+z}{1-z}\right) = \frac{z}{(1-z)^2}. \tag{2.12}$$

Cette équation est obtenue directement à partir de la récurrence (2.11), en la multipliant par z^n puis en sommant par rapport à n. Cette équation différentielle est alors résolue par la méthode de variation de la constante, et on obtient

$$S(z,u) = \frac{ze^{-uz}}{(1-z)^{2u}}\left(1 + \int_0^z (1-t)^{2(u-1)}e^{ut}dt\right). \tag{2.13}$$

Cette série génératrice fournit beaucoup d'informations sur les listes à sauts. Tout d'abord, $S_n'(1)$ est l'espérance de la longueur de cheminement interne, ensuite le théorème des quasi-puissances de Hwang [Hwa94] appliqué à un équivalent de $S_n(u)$ au voisinage de 1 montre que la distribution des $S_{n,k}$, pour k compris entre 0 et n est asymptotiquement gaussienne quand n tend vers l'infini.

Pour obtenir la lci, on commence par dériver S par rapport à u, puis on évalue la fonction obtenue en $u=1$. Le coefficient de z^n est ensuite obtenu par analyse de singularité, qu'on trouve dans [FO90]. On obtient alors la proposition suivante

Proposition 2.11 *La longueur de cheminement interne moyenne d'une liste à sauts de taille* n *est*

$$2n\log n + n(-3 - 2\mathrm{Ei}(1) + e^{-1}) + 2\log n - 2\mathrm{Ei}(1) + e^{-1} + 3 + o(1),$$

où γ *est la constante d'Euler, et* Ei *la fonction exponentielle intégrale de l'équation* (2.9).

Cette méthode s'applique également à l'équation de récurrence qui concerne le nombre moyen de comparaisons effectuées pour trouver un élément dans une liste à sauts, qui est

$$C_n(u) = u^3 + \frac{1}{n-1}\left(\sum_{k=1}^{n-1} uC_k(u) + u^2 C_{n-k-1}(u)\right). \tag{2.14}$$

Cette récurrence traduit le fait que pour décider si un élément est égal à la racine ou plus grand que le pointeur `jump`, ou strictement inférieur au pointeur `jump`, il faut soit une soit deux comparaisons. En effet la comparaison dont on dispose naturellement est binaire, et ne peut séparer les différents éléments qu'en deux catégories. La stratégie utilisée est que la première comparaison dit si l'élément recherché est plus grand que le pointeur `jump` (ce qui permet de poursuivre la recherche dans le sous-arbre droit), ou s'il est strictement inférieur. Dans ce second cas, une seconde comparaison dit si l'élément est plus grand que le pointeur `next` (sous-arbre gauche) ou enfin, s'il est égal à la racine.

Le nombre moyen de comparaisons binaires est alors

$$3n\log n + n(-3 - 3\mathrm{Ei}(1) + 3e^{-1}) + 3\log n - 3\mathrm{Ei}(1) + 3e^{-1} + 9/2 + o(1).$$

La preuve de ce développement est analogue à la preuve de la longueur interne de cheminement.

Loi limite des coefficients. La nature des coefficients $S_{n,k}$ pour $0 \leq k \leq n$ et lorsque n tend vers l'infini permet de se faire une idée du profil de l'arbre, c'est à dire savoir à quelle profondeur il y a le plus de nœuds, et quelle est la dispersion.

Par analyse de singularité de la fonction $S(z,u)$, dont l'expression est fournie par l'équation (2.13), on a

$$S_n(u) \sim \frac{e^{-u}(\lambda(u)+1)}{\Gamma(2u)} n^{2u-1} + O(n^{2u-2}), \tag{2.15}$$

uniformément dans un voisinage de $u = 1$, avec $\lambda(u) = \int_0^1 (1-t)^{2(u-1)} e^{ut} dt$. En utilisant le théorème des quasi-puissances de Hwang [Hwa94], on aboutit à la proposition suivante

Proposition 2.12 *La variable aléatoire X_n, niveau d'insertion, dont la loi est définie par $\mathrm{P}(X_n = k) = S_{n,k}/S_n(1)$ est asymptotiquement gaussienne lorsque n tend vers l'infini, avec une moyenne équivalente à $2\log n$ et une variance équivalente à $2\log n$*

Cette proposition montre que la profondeur des nœuds est concentrée autour de $2\log n$ et que les profils des arbres binaires de recherche et des arbres binaires associés aux listes à sauts sont similaires.

2.4.5 Hauteur

La hauteur d'un arbre est définie comme le maximum des profondeurs de tous les nœuds. De façon cohérente, on définit la hauteur d'une liste à sauts comme la hauteur de l'arbre binaire associé. On sait [Dev86, Drm01] que la hauteur moyenne d'un arbre binaire de recherche de taille n est équivalente à $c\log n$, où la constante c est définie par $\left(\frac{2e}{c}\right)^c = e$, et vaut numériquement $c = 4.311\ldots$.

Grâce aux similitudes entre listes à sauts et arbres binaires de recherche, on peut énoncer une proposition sur la hauteur moyenne des listes à sauts.

Proposition 2.13 *La hauteur moyenne d'une liste à sauts de taille n est bornée supérieurement par une quantité de la forme $c\log n + o(\log n)$, où $c \approx 4.3$ est définie par $\left(\frac{2e}{c}\right)^c = e$.*

Preuve. La preuve de cette proposition se fait par comparaison entre la hauteur moyenne d'un arbre binaire de recherche, et la hauteur moyenne d'une liste à sauts de même taille. On commence donc par préciser les notations respectives. On note A_n (A pour arbre) la variable aléatoire représentant la distribution de la hauteur des arbres binaires de recherche de taille n. On note ensuite J_n (J pour jumplist) la variable aléatoire qui code la hauteur des listes à sauts. La quantité $\Pr(J_n = k)$ est égale à la probabilité qu'une liste à sauts de taille n soit de hauteur k.

Pour un arbre binaire donné, qu'il soit de recherche ou issu d'une liste à sauts, la hauteur est égale au maximum de la hauteur de ses deux sous-arbres, plus 1. Ceci induit des récurrences sur les distributions de probabilité A_n et J_n, en notant que pour qu'un arbre soit de hauteur inférieure à h, il faut que ses deux fils soient de hauteur inférieure à $h-1$:

$$\Pr(A_n \leq k) = \frac{1}{n}\sum_{k=1}^{n} \Pr(A_{k-1} \leq h-1)\Pr(A_{n-k} \leq h-1),$$

et

$$\Pr(J_n \leq k) = \frac{1}{n-1}\sum_{k=1}^{n-1} \Pr(J_{k-1} \leq h-1)\Pr(J_{n-k} \leq h-1).$$

Les conditions initiales de ces récurrences sont données par le fait qu'un arbre de taille zéro a une hauteur égale à zéro dans tous les cas. Pour comparer les séquences A_n et J_n, on procède par récurrence. On suppose que pour tout $n < N$, on a la relation de domination stochastique $\Pr(J_n \leq h) \geq \Pr(A_n \leq h)$. On peut alors minorer $\Pr(J_N \leq h)$:

$$\begin{aligned}
n\Pr(J_N \leq h) &= \sum_{k=1}^{N} \Pr(J_{k-1} \leq h-1)\Pr(J_{N-k} \leq h-1) - \Pr(J_{N-1} \leq h-1) + \Pr(J_N \leq h) \\
&\geq \sum_{k=1}^{N} \Pr(J_{k-1} \leq h-1)\Pr(J_{N-k} \leq h-1) \\
&\geq \sum_{k=1}^{N} \Pr(A_{k-1} \leq h-1)\Pr(A_{N-k} \leq h-1) \\
&\geq n\Pr(A_N \leq h).
\end{aligned}$$

Le passage de la première à la deuxième ligne provient de l'inégalité $\Pr(J_{N-1} \leq h-1) \leq \Pr(J_N \leq h)$ qui exprime que lorsqu'on ajoute un nœud à un arbre, sa hauteur n'augmente pas de plus de 1.

L'inégalité obtenue sur les probabilités permet d'obtenir une inégalité sur les espérances car

$$\mathbb{E}A_n = \sum_h \Pr(A_n \geq h) \quad \text{et} \quad \mathbb{E}J_n = \sum_h \Pr(J_n \geq h).$$

La majoration $\Pr(J_N \leq h) \geq \Pr(A_N \leq h)$ entraîne la minoration $\Pr(J_N \geq h) \leq \Pr(A_N \geq h)$, et donc après sommation $\mathbb{E}J_n \leq \mathbb{E}A_n$. On a montré ainsi que la hauteur moyenne d'une liste à sauts est inférieure à la hauteur moyenne d'un ABR, ce qui prouve la proposition. ■

2.4.6 Profondeur du k-ième élément

La connaissance de la profondeur du k-ième élément permet d'obtenir de façon précise le temps moyen d'accès à chaque nœud de la liste. Cela permet de mieux comparer les listes à sauts et les

arbres binaires de recherche. Cette section utilise les résultats asymptotiques développés dans ce chapitre.

Cette analyse de la profondeur moyenne du k-ième élément dans une liste à sauts est à l'origine de mon intérêt pour l'asymptotique bivariée.

L'organisation de cette partie est la suivante. On montre tout d'abord une récurrence satisfaite par les coefficients $f_{n,k}$. Cette récurrence permet d'obtenir une comparaison entre listes à sauts et arbres binaires de recherche. Ensuite on obtient une expression pour la série génératrice des $f_{n,k}$. Cette série génératrice est alors séparée en plusieurs parties qu'on sait analyser, et on obtient un résultat asymptotique sur $f_{n,k}$ lorsque k n'est ni trop petit ni trop grand.

Récurrence

Si on note $f_{n,k}$ la profondeur moyenne du k-ième élément dans une liste à sauts de taille n, on peut écrire la récurrence suivante, pour k supérieur à 2.

$$f_{n,k} = 1 + \frac{1}{n-1}\sum_{i=2}^{k-1} f_{n-i+1,k-i+1} + \frac{1}{n-1}\sum_{i=k+1}^{n} f_{i-2,k-1}, \tag{2.16}$$

avec comme condition initiale $f_{n,1} = 0$.

Cette récurrence se déduit de la description récursive des listes à sauts. En effet le pointeur `jump` issu de 1 atterrit sur le i-ième élément, $2 \leq i \leq n$ avec probabilité uniforme, c'est-à-dire $\frac{1}{n-1}$. Pour tous les éléments excepté le premier, il faut suivre au moins un pointeur, ce qui explique le "1" dans la récurrence. Ensuite il y a deux possibilités, qui sont chacune prises en compte par une somme. Soit le premier pointeur `jump` arrive avant l'élément recherché, auquel cas on utilise ce pointeur `jump` et on doit chercher l'élément numéro $k-i+1$ dans une liste à sauts de taille $n-i+1$ (qui est celle qui suit le pointeur `jump`) ; soit le pointeur `jump` arrive après l'élément recherché auquel cas on doit commencer par marcher d'un pas, puis rechercher l'élément numéro $k-1$ dans une liste à sauts de taille $i-2$. On a alors obtenu exactement la récurrence (2.16).

Majoration

Il est possible, grâce à la récurrence (2.16) de comparer les listes à sauts aux ABR, et de montrer la propriété suivante.

Propriété 2.1 *Soit $f_{n,k}$ la profondeur moyenne du k-ième élément d'une liste à sauts, et $g_{n,k}$ la profondeur moyenne du k-ième élément d'un ABR. Alors*

$$f_{n,k} \leq \frac{n}{n-1} g_{n,k}.$$

Pour les arbres binaires de recherche, la récurrence pour la séquence $g_{n,k}$ est (comme vu dans la section 2.3.3)

$$g_{n,k} = \frac{1}{n}\sum_{i=1}^{k-1}(1 + g_{n-i,k-i}) + \frac{1}{n}\sum_{i=k+1}^{n}(1 + g_{i-1,k})$$

qui peut se récrire de façon à ressembler à la récurrence des listes à sauts comme

$$g_{n,k} = \frac{n-1}{n} + \frac{1}{n}\sum_{i=2}^{k} g_{n-i+1,k-i+1} + \frac{1}{n}\sum_{i=k+1}^{n} g_{i-1,k}.$$

D'après la proposition 2.8, on sait que $g_{n,k} = H_k + H_{n-k}$, où H_k désigne le k-ième nombre harmonique.

Pour comparer les deux séquences $f_{n,k}$ et $g_{n,k}$ qui correspondent respectivement aux listes à sauts et aux ABR, on introduit une troisième séquence, intermédiaire entre les deux précédentes, qui satisfait la récurrence

$$h_{n,k} = \frac{n-1}{n} + \frac{1}{n}\sum_{i=2}^{k-1} h_{n-i+1,k-i+1} + \frac{1}{n}\sum_{i=k+1}^{n} h_{i-2,k-1}, \tag{2.17}$$

avec les conditions initiales $h_{n,1} = g_{n,1}$.

On compare maintenant $h_{n,k}$ à $g_{n,k}$. Le but est de montrer que pour tout n et k on a $h_{n,k} \leq g_{n,k}$, ce qu'on prouve par récurrence. L'hypothèse de récurrence est que pour tout $N < n$ on a $h_{N,k} \leq g_{N,k}$. La valeur $h_{n,k}$ définie par l'équation (2.17) peut être majorée grâce à l'hypothèse de récurrence par

$$h_{n,k} \leq \frac{n-1}{n} + \frac{1}{n}\sum_{i=2}^{k} g_{n-i+1,k-i+1} + \frac{1}{n}\sum_{i=k+1}^{n} g_{i-2,k-1}.$$

Grâce au résultat $g_{n,k} = H_k + H_{n-k}$, on voit que $g_{i-2,k-1} = g_{i-1,k} - \frac{1}{k}$, donc

$$h_{n,k} \leq \frac{n-1}{n} + \frac{1}{n}\sum_{i=2}^{k} g_{n-i+1,k-i+1} + \frac{1}{n}\sum_{i=k+1}^{n} g_{i-1,k} = g_{n,k}.$$

Cela prouve que quel que soit n et k on a $h_{n,k} \leq g_{n,k}$.

On continue en comparant $h_{n,k}$ et $f_{n,k}$. De par leur définition, on a

$$f_{n,k} = \frac{n}{n-1} h_{n,k}.$$

Ce qui donne finalement

$$f_{n,k} \leq \frac{n}{n-1} g_{n,k} = \frac{n}{n-1}(H_k + H_{n-k}) \tag{2.18}$$

Remarque. On constate facilement que $f_{n,k} \leq k$, car chaque élément est accessible par une succession de pointeurs `next` de longueur k. Ceci permet de raffiner la majoration précédente lorsque k est petit.

Asymptotique

On obtient un résultat asymptotique pour $f_{n,k}$ grâce aux méthodes présentées précédemment, si k n'est ni trop grand ni trop petit, et l'on peut alors constater que la majoration obtenue dans l'équation (2.18) est très proche de la réalité. On énonce de fait le théorème suivant :

Théorème 2.1 *Si $f_{n,k}$ désigne la profondeur moyenne du k-ième élément dans une liste à sauts de taille n, alors on a*

$$f_{n,k} \sim \log(n-k) + \log k$$

pour $\log^2 n \leq k \leq n - \log^2 n$, lorsque n tend vers l'infini.

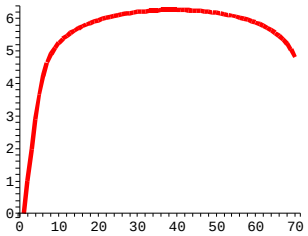

FIG. 2.11 – La courbe des $f_{n,k}$, pour $n = 70$

Pour la zone concernée par ce théorème, la majoration énoncée dans la proposition 2.1 est effectivement précise. On constate expérimentalement que lorsque k est en dehors de cette zone, cette majoration reste valable (à condition de tenir compte en plus de la majoration $f_{n,k} \leq k$). Pour k petit, ceci s'explique intuitivement par le fait que en moyenne le premier pointeur `jump` atterrit en $n/2$, le deuxième, toujours en moyenne en $n/4$, et ainsi de suite pour les premier sauts. Ceci revient à dire que les premiers éléments sont généralement atteints en marchant, et donc que $f_{n,k} \sim k$ si k est "petit". En pratique on constate que cette équivalence est valable presque jusqu'à $\log n$. Pour k grand, on observe de surcroît que la profondeur moyenne tend à diminuer jusqu'à valoir $\log n$. Une explication intuitive du fait que cette profondeur décroît est que vers la fin de la liste à sauts, il y a une forte concentration d'arrivées de pointeurs, ce qui permet donc d'arriver plus rapidement à des éléments situés vers la fin. La figure 2.11 résume ce paragraphe en montrant l'exemple de la courbe des $f_{n,k}$ pour $n = 70$.

La preuve du théorème 2.1 occupe la fin de cette sous section. On commence par expliciter la série génératrice $F(z,u)$ de $f_{n,k}$ pour ensuite extraire l'asymptotique des coefficients de cette série génératrice. Pour cela on par énonce un lemme préliminaire qui prend en compte les termes du type $(1-z)^u$ qui y apparaissent. La série génératrice F s'exprime comme une somme de trois termes. Intuitivement le premier correspond à la singularité située en $z = 1$, le deuxième à celle en $z = 1/u$, et le troisième au reste. On traite séparément chacun des trois termes afin d'obtenir le résultat recherché.

La série génératrice. La série génératrice ordinaire de la séquence $f_{n,k}$, définie par $F(z,u) = \sum_{n,k} f_{n,k} u^k z^n$, satisfait l'équation différentielle

$$zF'_z(z,u) - F(z,u)\left(1 - \frac{z^2u}{1-z} + \frac{zu}{1-zu}\right) = \frac{z^2u^2}{(1-u)(1-z)^2} - \frac{u^3z^2}{(1-u)(1-zu)^2}.$$

Cette équation se déduit de la récurrence satisfaite par $f_{n,k}$. La méthode de variation de la constante fournit la solution explicite

$$F(z,u) = \frac{ze^{-uz}}{(1-z)^u(1-zu)} \int_0^z (1-t)^u(1-tu)e^{ut}\frac{u^2}{1-u}\left(\frac{1}{(1-t)^2} - \frac{u}{(1-tu)^2}\right) dt. \qquad (2.19)$$

Un lemme préliminaire. Par rapport aux résultats énoncés dans la section 2.2, on constate que la série génératrice F est beaucoup plus complexe. Les deux difficultés principales sont d'une part la présence de l'intégrale, et d'autre part le terme $(1-z)^u$, où la variable u apparaît désormais en exposant. Ce paragraphe aplanit cette deuxième difficulté.

Lemme 2.3 *Le coefficient de* $[z^n][u^k]$ *dans la fonction* $w(z,u) := \left(\frac{1}{1-z}\right)^u \frac{1}{1-zu} \log^i\left(\frac{1}{1-z}\right)$ *vérifie*

$$[z^n][u^k]\left(\frac{1}{1-z}\right)^u \frac{1}{1-zu}\log^i\left(\frac{1}{1-z}\right) \sim \log^i(n-k) \tag{2.20}$$

pour k *tel que* $\log^2 n \leq k \leq n - \log^2 n$, *lorsque* n *tend vers l'infini.*

Preuve. La preuve reprend la preuve de la proposition 2.3 où sont extraits des coefficients bivariés par double contour de Hankel. La partie principale de ce double contour de Hankel est celle qui se situe autour de $z = 1$ et de $u = 1$, et est notée $\Gamma_{1,2,3} \times \Delta_{1,2,3}$ dans la preuve de ce théorème. Sur cette partie principale du contour double, on peut approcher $w(z,u)$ et écrire

$$w(z,u) \sim \frac{1}{1-z}\frac{1}{1-zu}\log^i\left(\frac{1}{1-z}\right)\left(1 + O\left((1-u)\log\frac{1}{1-z}\right)\right).$$

La preuve de l'énoncé se conclut alors de manière identique à celle de la proposition 2.3. ■

On voit que si la fonction $w(z,u)$ du lemme 2.3 est multipliée par une fonction analytique $a(z,u)$ telle que $a(1,1) = 1$, alors l'estimation du membre droit de (2.20) reste valable.

Remarque. On obtient de la même manière que

$$[z^n u^k]\left(\frac{1}{1-z}\right)^u \frac{1}{1-zu}\log^i\left(\frac{1}{1-zu}\right) \sim \log^i k.$$

L'extraction bivariée pour une série génératrice simplifiée. Pour commencer, on s'intéresse à une série génératrice plus simple qui est

$$G(z,u) = \frac{1}{(1-z)^u(1-zu)}\int_0^z (1-t)^u(1-tu)\frac{u^2}{1-u}\left(\frac{1}{(1-t)^2} - \frac{u}{(1-tu)^2}\right)dt.$$

L'analyse de cette série génératrice est le cœur du problème, et on le traite par conséquent en détail.

La simplification provient du fait que les termes exponentiels (qui sont analytiques en $(1,1)$) ont été enlevés. On vise à approcher le coefficient de $[z^n u^k]$ de $G(z,u)$, qu'on note $g_{n,k}$. Cette série génératrice se récrit sous une forme plus compacte, qui permet de mettre en relief les singularités

$$G(z,u) = \frac{1}{(1-z)^u}\frac{1}{1-zu}\int_0^z \frac{(1-t)^{u-2}(1-t^2u)}{(1-tu)}dt,$$

dont on note l'intégrande $g(t,u)$:

$$g(t,u) := \frac{(1-t)^{u-2}(1-t^2u)}{(1-tu)}.$$

Pour extraire l'asymptotique des coefficients $g_{n,k}$ en appliquant la méthode de double contour de Hankel, il faut gérer le problème de l'intégrale. Pour cela on sépare l'intégrande en deux parties $g(t,u) = g(t,1) + (g(t,u) - g(t,1))$. Le premier terme s'intègre sans problème, et le deuxième est étudié ensuite.

La contribution du premier terme s'écrit

$$\frac{1}{(1-z)^u}\frac{1}{1-zu}\int_0^z g(t,1)dt = \frac{1}{(1-z)^u}\frac{1}{1-zu}\left(-z + 2\log\frac{1}{1-z}\right)$$

D'après le lemme préliminaire, le coefficient de $[z^n u^k]$ dans cette fonction est équivalent à $2\log(n-k)$.

Pour la deuxième partie, l'idée est d'extraire la partie correspondant à la singularité $tu = 1$. Ceci donne un deuxième terme nommé $h(t,u)$ égal à $\frac{-t(1-u)(1-t)^{u-2}}{1-tu}$. Ensuite on nomme $r(t,u)$ le reste de la fonction, c'est à dire

$$g(t,u) = g(t,1) + h(t,u) + r(t,u).$$

L'étude de l'intégrale est donc séparée en trois termes, dont deux principaux, à savoir $g(t,1)$ (dont la contribution est déjà calculée) et $h(t,u)$. Enfin le troisième terme $r(t,u)$ est un terme d'erreur.

• Une approximation de h. La fonction $h(t,u)$ est égale à $h(t,u) := \frac{-t(1-u)(1-t)^{u-2}}{1-tu}$. Sa contribution est calculée via l'approximation $h(t,u) \sim \frac{u-1}{(1-t)(1-tu)}$, valable au voisinage de $(1,1)$, qui est intégrable :

$$\int_0^z \frac{u-1}{(1-t)(1-tu)} = -\log\frac{1}{1-z} + \log\frac{1}{1-zu}. \tag{2.21}$$

Cette approximation est justifiée car la différence s´écrit

$$h(t,u) - \frac{u-1}{(1-t)(1-tu)} = \frac{u-1}{(1-t)(1-tu)}\left((1-t)^{u-1} - 1\right). \tag{2.22}$$

Pour le domaine $1 - z \geq 1/n$ et $1 - u \geq 1/k$, c'est-à-dire qu'on se place dans les conditions adéquates pour pouvoir ultérieurement extraire le coefficient $[z^n u^k]$, on dispose de l'estimation $\left((1-t)^{u-1} - 1\right) = O((1-u)\log\frac{1}{1-t})$. La différence (2.22) est alors bornée par

$$h(t,u) - \frac{u-1}{(1-t)(1-tu)} = O\left(\frac{1}{(1-t)(1-tu)}(1-u)^2\log\frac{1}{1-t}\right)$$

uniformément pour t entre 0 et z. Grâce à une décomposition en éléments simples, on obtient que l'intégrale de cette différence vaut

$$\int_0^z h(t,u) - \frac{u-1}{(1-t)(1-tu)} = o\left(\int_0^z \frac{u-1}{(1-t)(1-tu)}\right).$$

La contribution de h est donc équivalente à la contribution du terme de l'équation (2.21).

• La contribution de h. La contribution de h est égale à

$$[z^n u^k]\frac{1}{(1-z)^u(1-zu)}\int_0^z h(t,u) \sim [z^n u^k]\frac{1}{(1-z)^u(1-zu)}\left(-\log\frac{1}{1-z} + \log\frac{1}{1-zu}\right).$$

Par le lemme 2.3, on obtient que la partie droite de l'équivalence ci dessus est égale à $\log k - \log(n-k)$.

• La contribution du reste r. Le reste r s'écrit sous la forme $r(t,u) = \frac{1+t}{1-t}((1-t)^{u-1} - 1)$. Cette fonction s'intègre de façon explicite par rapport à t, et on obtient

$$\int_0^z r(t,u)dt = -2\log\frac{1}{1-z} + z + \frac{1}{1-u}\left((1-z)^{u-1}(1/u + 1 + z - z/u) - (1 + 1/u)\right).$$

La contribution de ce terme d'erreur qui est $[z^n u^k]\frac{1}{(1-z)(1-zu)}\int_0^z r(t,u)dt$ est alors $O(1)$ en vertu du théorème 2.3 et du lemme 2.3.

A ce stade, on a établi le résultat recherché pour une série génératrice simplifiée.

La preuve complète. Pour la série génératrice complète, on revient à la série génératrice bivariée du coût du k-ième élément dans une liste à sauts de taille n, énoncée dans l'équation (2.19), et qu'on rappelle ici.

$$F(z,u) = \frac{ze^{-uz}}{(1-z)^u(1-zu)} \int_0^z (1-t)^u(1-tu)e^{ut}\frac{u^2}{1-u}\left(\frac{1}{(1-t)^2} - \frac{u}{(1-tu)^2}\right) dt.$$

On constate que par rapport à la série génératrice simplifiée $G(z,u)$ qu'on vient d'étudier, il y a une différence : les facteurs u^2e^{ut} et ze^{uz} qui sont respectivement à l'intérieur et à l'extérieur de l'intégrale. De fait il apparaît que cette différence n'influe pas sur l'asymptotique trouvée pour la série génératrice simplifiée.

Les deux facteurs supplémentaires ajoutés sont analytiques, et donc vont contribuer seulement par leur valeur en la singularité $(1,1)$. On note qu'au voisinage de $u=1$, on a $u^2e^{ut} \sim e^t$. Cette équivalence peut ensuite être injectée aux différentes étapes de la preuve précédente et on constate que cela modifie les valeurs précédemment trouvées pour G d'une constante multiplicative e.

Le cas de ze^{-uz} qui figure à l'extérieur de l'intégrale est plus simple. Comme au voisinage de $(z,u)=(1,1)$, la fonction ze^{-uz} est équivalente à e^{-1}, on peut vérifier que la modification due à cet ajout est un facteur e^{-1}.

Ainsi, les deux modifications (e et e^{-1}) se contrebalancent, de sorte qu'on obtient $[z^nu^k]F(z,u) \sim [z^nu^k]G(z,u) \sim \log k + \log(n-k)$. Ceci conclut la preuve du théorème 2.1.

Conclusion

On a montré dans la partie Majoration de cette section une majoration du coût moyen d'accès au k-ième élément dans une liste à sauts de taille n notée $f_{n,k}$ qui est

$$f_{n,k} \leq \min\{k, \frac{n}{n-1}(H_k + H_{n-k})\},$$

pour tout k.

Le paragraphe Asymptotique de cette section montre que cette majoration est aussi un équivalent asymptotique lorsque k est central, c'est à dire entre $\log^2 n$ et $n - \log^2 n$. L'expérience montre qu'en fait cette majoration colle étroitement à la réalité sur tout l'intervalle $[0,n]$.

En conclusion les listes à sauts et les ABR ont des coûts d'accès extrêmement similaires, excepté pour les petites valeurs où les listes à sauts sont nettement meilleures. Ceci est un avantage dans le cas de tels éléments seraient souvent recherchés.

Pour le cas général ou les éléments ont une probabilité d'accès uniforme ou tout simplement inconnue, le nombre moyen de pointeurs à suivre est du même ordre de grandeur pour les deux structures, tout en étant légèrement inférieur pour les listes à sauts.

2.5 Deux singularités confluentes

La section 2.2 était consacrée à l'extraction bivariée de coefficients de fonctions du type $(1-z)^{-a}(1-zu)^{-b}$. La réussite de l'extraction par double contour de Hankel repose sur le fait que, si cette série possède deux singularités en z, il n'y a en revanche qu'une singularité en u. La méthode développée à cette occasion consiste à contourner la difficulté en commençant par calculer le coefficient de u^k, puis celui de z^n. Ce dernier calcul a alors lieu en univarié sans problème de confluence de singularité. Par contre, pour calculer le coefficient de z^n pour tout u de la fonction $(1-z)^a(1-zu)^b$, il faut affronter ce problème de double singularité dont la distance n'est pas a priori

minorée. La méthode introduite ici consiste à utiliser la transformation de Mellin afin de "séparer" les deux singularités.

Cette technique est de fait très générale, elle permet ensuite d'extraire les coefficients de fonctions présentant une confluence de singularités du même type (situées éventuellement en d'autres points que 1 et $1/u$).

Le résultat principal est énoncé ci-dessous. La première étape de preuve revient à d'exprimer le coefficient de z^n par une intégrale de Cauchy, la seconde à calculer la transformée de Mellin de l'expression obtenue, pour finalement obtenir le résultat recherché par la transformée de Mellin inverse.

2.5.1 Résultats asymptotiques

Théorème 2.2 *Le coefficient de* $[z^n]$ *dans* $\frac{1}{(1-z)^a}\frac{1}{(1-zu)^b}$, *avec* u *au voisinage de* 1 *vérifie*

$$[z^n]\frac{1}{(1-z)^a}\frac{1}{(1-zu)^b} = \overline{f}_n(h) = \sum_i n^{a+b+i-1}\frac{1}{\Gamma(a+b+i)}\frac{\Gamma(b+i)}{\Gamma(b)}\frac{1}{i!}h^i\left(1+O\left(\frac{\log^2 n}{n}\right)\right). \tag{2.23}$$

où $u = 1+h$, *avec* $|h| = O\left(\frac{\log^2 n}{n}\right)$.

Remarque. Ce résultat reste vrai si $f(z,u)$ est multiplié par une fonction analytique $w(z,u)$ telle que $w(1,1) = 1$.

Preuve.

La preuve du théorème 2.2 se fait en deux parties. Tout d'abord, on exprime le coefficient de $[z^n]$ grâce au théorème de Cauchy, puis on calcule cette intégrale grâce à la transformée de Mellin. Le résultat obtenu est alors une série hypergéométrique, qui peut être utilisée efficacement pour obtenir le coefficient de u^k. Cette dernière étape est effectuée dans la section 2.6.

Remarque. Tout d'abord pour éclairer le contenu de ce théorème, on peut l'adapter au cas particulier $a = 1$ et $b = 1$. On doit calculer le coefficient $[z^n]\frac{1}{(1-z)}\frac{1}{(1-zu)}$, qui est s'évalue, par un calcul élémentaire à $\frac{1-u^{n+1}}{1-u}$. Or l'estimation fournie par le théorème précédent est $n\sum_i \frac{h^i}{(i+1)!}$ soit $\frac{n(e^h-1)}{h}$, ce qui est équivalent au résultat attendu car $u^n = (1+h/n)^n \sim e^h$.

Intégrale de Cauchy. On rappelle que $f(z,u)$ est ici définie par $f(z,u) := \frac{1}{(1-z)^a}\frac{1}{(1-zu)^b}$. Le coefficient de $[z^n]$ est noté $f_n(u) := [z^n]f(z,u)$. La fonction $f_n(u)$ s'exprime grâce au théorème de Cauchy

$$f_n(u) = \frac{1}{2i\pi}\oint f(z,u)\frac{dz}{z^{n+1}}.$$

On opère alors les changements de variables suivants qui permettent de se ramener à un voisinage des points 1 et $1/u$, qui sont les singularités en z, et l'on se place dans la bonne échelle pour z, c'est à dire qu'on pose

$$z = 1+\frac{t}{n} \qquad \text{et} \qquad u = 1+h.$$

On peut alors récrire $f_n(u)$ de la façon suivante, en définissant $\overline{f}_n(h) := f_n(1+h)$,

$$f_n(u) = \overline{f}_n(h) = \frac{1}{2i\pi}\oint\left(\frac{-n}{t}\right)^a\frac{(-1)^b}{(t/n+h+th/n)^b}\left(1+\frac{t}{n}\right)^{-n-1}\frac{dt}{n}$$

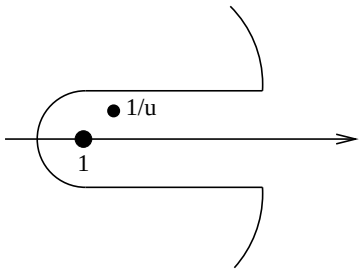

FIG. 2.12 – Un contour de Hankel, avec deux singularités très proches

Avant d'estimer cette intégrale, on précise quelques notations. L'intégrale selon z est calculée selon un contour de Hankel de Hankel noté $\Gamma_{1,2,3,4}$ en reprenant les notations de la section 2.2.1. Ce contour passe autour de $z=1$, à une distance $O\left(\frac{\log^2 n}{n}\right)$ $(\Gamma_{1,2,3})$, puis autour de $z=0$ à une distance $1+O\left(\frac{\log^2 n}{n}\right)$ (Γ_4). La distance du contour $\Gamma_{1,2,3}$ à la singularité 1 est suffisante pour laisser la singularité $1/u$ à l'extérieur de ce contour. On est alors bien dans le cadre du théorème de Cauchy, car seule la singularité située en 0 est à l'intérieur du contour $\Gamma_{1,2,3,4}$.

La figure 2.12 montre la partie principale du contour de Hankel d'intégration pour le cas de figure étudié, où u est très proche de 1. (Pour que le contour soit complet, il faut refermer le cercle). On voit que la singularité $1/u$ est à l'intérieur du "tube" que le contour dessine autour de la singularité 1. Cela fait qu'il ne suffit pas seulement de considérer la singularité 1 pour approximer la valeur de l'intégrale le long de ce tube, et que les méthodes classiques d'analyse de singularité ne s'appliquent pas.

En utilisant encore les notations de la section 2.2.1, on considère d'une part la partie principale de l'intégrale qui est sur le contour $\Gamma_{1,2,3}$ (le tube), et d'autre part la partie selon Γ_4 qui est traitée plus tard en terme d'erreur.

On s'intéresse donc pour l'instant à l'intégrale principale

$$\overline{f}_{1,n}(h)=\frac{1}{2i\pi}\int_{\Gamma_{1,2,3}}\left(\frac{-n}{t}\right)^a\frac{(-1)^b}{(t/n+h+th/n)^b}\left(1+\frac{t}{n}\right)^{-n-1}\frac{dt}{n}$$

sur laquelle on dispose des équivalences suivantes : $\left(1+\frac{t}{n}\right)^{-n-1}\sim e^{-t}$ car t est petit devant n ; $t/n+h+th/n=(t/n+h/n)(1+O(\log^2 n/n))$, car on a $|t|\leq\log^2 n$. L'intégrale peut alors se récrire

$$\overline{f}_{1,n}(h)\sim\frac{n^{a-1+b}}{2i\pi}\int_{\Gamma_{1,2,3}}(-t)^{-a-b}\frac{1}{(1+\frac{hn}{t})^b}e^{-t}\frac{dt}{n}=:\overline{f}_{2,n}(h).$$

La suite est consacrée à l'étude de $\overline{f}_{2,n}(h)$

Transformée de Mellin. L'intégrale $\overline{f}_{2,n}(h)$ ne peut pas être évaluée directement, à cause de ce problème de double singularité. On utilise alors la transformée de Mellin afin d'obtenir un développement asymptotique de $\overline{f}_{2,n}(h)$ en puissances croissantes de h.

La transformée de Mellin de $\overline{f}_{2,n}(h)$ est notée $f_n^*(s)$. Par définition, on a

$$f_n^*(s):=\int_0^\infty\overline{f}_{2,n}(h)h^{s-1}dh.$$

Les deux intégrales peuvent être interverties, ce qui permet d'écrire

$$f_n^*(s) = \frac{n^{a+b-1}}{2i\pi} \int_{\Gamma_{1,2,3}} (-t)^{-a-b} e^{-t} \left(\int_0^\infty \frac{1}{(1+\frac{hn}{t})^b} h^{s-1} dh \right) dt. \tag{2.24}$$

L´intégrale selon h peut être évaluée grâce aux propriétés classiques de la transformation de Mellin [FGD95], dont l'essentiel a été rappelé à la section 1.2. En particulier, on sait que la transformée de Mellin de $\frac{1}{(1+x)^b}$ est $\frac{\Gamma(b-s)\Gamma(s)}{\Gamma(b)}$ (voir [FS05]), de sorte que par un simple changement de variable on obtient

$$\int_0^\infty \frac{1}{(1+\frac{hn}{t})^b} h^{s-1} dh = (t/n)^s \frac{\Gamma(b-s)\Gamma(s)}{\Gamma(b)}.$$

La substitution de ce résultat dans l'équation (2.24) entraîne

$$f_n^*(s) = \frac{n^{a+b-s-1}}{2i\pi} \int_{\Gamma_{1,2,3}} (-t)^{s-a-b} e^{-t} dt \; \frac{\Gamma(b-s)\Gamma(s)}{\Gamma(b)} (-1)^s.$$

L'intégrale ci-dessus est semblable à celles manipulées lors de la section 2.2.1 à propos de l'extraction de coefficients par contour de Hankel. On peut alors approximer cette intégrale, et obtenir l'estimation

$$f_n^*(s) \sim n^{a+b-s-1} (-1)^s \frac{1}{\Gamma(a+b-s)} \frac{\Gamma(b-s)\Gamma(s)}{\Gamma(b)},$$

asymptotiquement lorsque n tend vers l'infini.

Transformation de Mellin inverse. La connaissance des singularités de la transformée de Mellin permet d'accéder à l'asymptotique de la fonction initiale. On étudie pour cela $f_n^*(s)$ au voisinage de ses singularités. La bande fondamentale de cette transformée de Mellin est $\langle 0, b \rangle$(voir section 1.2), et ses singularités à gauche de la bande fondamentale sont situées aux entiers négatifs, car ce sont les pôles de la fonction $\Gamma(s)$. Un équivalent de $f_n^*(s)$ au voisinage d'un entier négatif $-i$ est :

$$f_n^*(s) \sim_{s \to -i} n^{a+b+i-1} \frac{1}{\Gamma(a+b+i)} \frac{\Gamma(b+i)}{\Gamma(b)} \frac{1}{i!} \frac{1}{s+i}.$$

Par les propriétés d'inversion de Mellin, le coefficient de h^i dans $\overline{f}_{2,n}$ est $n^{a+b+i-1} \frac{1}{\Gamma(a+b+i)} \frac{\Gamma(b+i)}{\Gamma(b)} \frac{1}{i!}$. Ceci nous permet de déduire que la fonction $\overline{f}_{2,n}(h)$ possède un développement en 0 de la forme

$$\overline{f}_{2,n}(h) \sim \sum_i n^{a+b+i-1} \frac{1}{\Gamma(a+b+i)} \frac{\Gamma(b+i)}{\Gamma(b)} \frac{1}{i!} h^i.$$

Ce développement en série entière de $\overline{f}_{2,n}$ implique

$$\overline{f}_{1,n} \sim n^{a+b-1} \sum_i \frac{n^i}{\Gamma(a+b+i)} \frac{\Gamma(b+i)}{\Gamma(b)} \frac{1}{i!} h^i, \tag{2.25}$$

lorsque n tend vers l'infini.

Terme d'erreur Γ_4. Les hypothèses prises concernant la position de u permettent de garantir que lorsque z est sur la partie Γ_4 de son contour d'intégration, on a $(1-zu)^{-1} \leq n$. Cela permet de reprendre les majorations classiques, et de montrer que, le long de Γ_4, l'intégrale est en $O(e^{-\log^2 n} n^{a+b})$, et est donc un terme d'erreur par rapport à l'intégrale le long de $\Gamma_{1,2,3}$. Cela entraîne

$$f_n(h) = \overline{f}_{1,n} + O(e^{-\log^2 n} n^{a+b}). \tag{2.26}$$

Conclusion. Les équations (2.26) et (2.25) entraînent l'estimation

$$\overline{f}_n(h) = \sum_i n^{a+b+i-1} \frac{1}{\Gamma(a+b+i)} \frac{\Gamma(b+i)}{\Gamma(b)} \frac{1}{i!} h^i + o(1), \tag{2.27}$$

uniformément pour $|h| = O\left(\frac{\log^2 n}{n}\right)$, lorsque n tend vers l'infini.

La preuve du théorème 2.2 est alors terminée.

■

2.5.2 Si u n'est pas proche de 1

Lorsque u n'est pas voisin de 1, c'est à dire que $1 - u \gg 1/n$, on peut se ramener à l'analyse de singularité classique pour calculer $f_n(u)$. Il y a trois cas de figure, selon que l'une des deux singularités domine, ou qu'elles sont en concurrence. On considère que 1 est la singularité dominante si $|1/u| > 1 + \frac{\log^2 n}{n}$, de même on dit que $1/u$ est la singularité dominante si $1 > |1/u| + \frac{\log^2 n}{n}$. Dans le cas intermédiaire on considère les points 1 et $1/u$ comme étant en concurrence. Les deux premiers cas se traitent assez facilement car on peut obtenir l'asymptotique de $f_n(u)$ grâce à un simple contour de Hankel autour de la singularité dominante. Pour le troisième cas, la méthode consiste à intégrer selon un contour de Hankel qui entoure les deux singularités séparément, et prend ainsi en compte les deux contributions.

Cas 1 : Si la singularité 1 domine alors

$$f_n(u) \sim \frac{1}{(1-u)^b} \frac{n^{a-1}}{\Gamma(a)}.$$

Cas 2 : Si la singularité $1/u$ domine alors

$$f_n(u) \sim \frac{1}{(1-1/u)^a} \frac{n^{b-1}}{\Gamma(b)} u^n.$$

Cas 3 : Si les deux singularités contribuent alors

$$f_n(u) \sim \frac{1}{(1-1/u)^a} \frac{n^{b-1}}{\Gamma(b)} u^n + \frac{1}{(1-u)^b} \frac{n^{a-1}}{\Gamma(a)}.$$

Les preuves de ces différentes équivalences sont une simple reprise du cas où il n'y a qu'une singularité. Le cas 3 étant l'addition des contributions des deux singularités, on remarque sans grande surprise que l'équivalent fourni dans le cas 3 est l'addition des équivalents fournis pour les cas 1 et 2.

2.6 Trois singularités confluentes

Le but de cette section est d'étudier la série génératrice bivariée

$$f(z,u) = \left(\frac{1}{1-z}\right)^a \left(\frac{1}{1-zu}\right)^b \left(\frac{1}{1-u}\right)^c, \tag{2.28}$$

avec éventuellement l'ajout de termes logarithmiques, et d'en extraire le coefficient de $[u^k z^n]$. On suppose toujours que k , $n-k$ et n sont proportionnels, et que $k < n$.

Cette fonction possède trois courbes singulières $u = 1$, $z = 1$ et $zu = 1$, qui se croisent au point $(1, 1)$, et qui vont toutes les trois participer à l'obtention du résultat final.

Le résultat asymptotique obtenu dans cette section se situe dans le prolongement naturel des résultats obtenus dans la section 2.2.2. Il permet d'étudier en toute généralité toute une famille de problèmes combinatoires, dont quelques exemple sont présentés à la fin de ce chapitre.

L'idée d'utiliser les propriétés des fonctions hypergéométriques pour obtenir l'asymptotique des coefficients bivariés m'a été suggérée par Michael Drmota, lors d'une visite que j'ai effectuée à Vienne.

2.6.1 Énoncés

Le théorème 2.3 explicite l'asymptotique des coefficients $f_{n,k} = [z^n u^k] f(z, u)$, avec k, $n-k$ et n proportionnels, et $k < n$.

Théorème 2.3 *Soit* $f(z,u) = \left(\frac{1}{1-z}\right)^a \left(\frac{1}{1-zu}\right)^b \left(\frac{1}{1-u}\right)^c$, *avec* $\Re(a) > 0$, $\Re(b) > 0$ *et* $\Re(c) > 0$, *alors*

$$[z^n u^k] f(z,u) \sim \frac{k^{b+c-1} n^{a-1}}{\Gamma(a)} \frac{1}{\Gamma(b+c)} {}_2F_1\left(1-a, b; b+c; \frac{k}{n}\right), \tag{2.29}$$

Théorème 2.4 *Soit la fonction* $g(z,u) = w(z,u) \left(\frac{1}{1-z}\right)^a \left(\frac{1}{1-zu}\right)^b \left(\frac{1}{1-u}\right)^c$, *où* $w(z,u)$ *est une fonction analytique, telle que* $w(1,1) = 1$, *et avec* $\Re(a) > 0$, $\Re(b) > 0$ *et* $\Re(c) > 0$, *alors*

$$[z^n u^k] g(z,u) \sim \frac{k^{b+c-1} n^{a-1}}{\Gamma(a)} \frac{1}{\Gamma(b+c)} {}_2F_1\left(1-a, b; b+c; \frac{k}{n}\right), \tag{2.30}$$

La suite de cette section est consacrée à la preuve de ces théorèmes. On commence par rappeler la définition des fonctions hypergéométriques. Le théorème 2.3 est prouvé dans la sous-section 2.6.3.

2.6.2 Fonctions hypergéométriques

Une suite f_n est dite hypergéométrique si elle satisfait une récurrence linéaire d'ordre 1 dont les coefficients sont des fractions rationnelles en n. Ces suites peuvent toujours être représentées comme des produits et quotients de valeurs de la fonction Γ en des points qui sont des fonctions linéaires en n.

Une série génératrice $f(z) = \sum f_n z^n$ est *hypergéométrique* si la suite de ses coefficients f_n est hypergéométrique. On note $\underline{a} = (a_0, \ldots, a_p)$ et $\underline{b} = (b_0, \ldots, b_q)$. Une fonction hypergéométrique peut donc se mettre sous la forme

$${}_pF_q(\underline{a}; \underline{b}; z) = \sum_{k=0}^{\infty} \prod_{i=1}^{p} \frac{\Gamma(a_i + k)}{\Gamma(a_i)} \prod_{j=1}^{q} \frac{\Gamma(b_i)}{\Gamma(b_i + k)} \frac{z^k}{k!}$$

Ces fonctions hypergéométriques ${}_2F_1$ vérifient deux propriétés, dont la première est due à Euler, et dont on se sert par la suite :

Proposition 2.14 *Si* $\Re(c) > \Re(b) > 0$, *alors*

$$ {}_2F_1(a,b;c;z) = \frac{\Gamma(c)}{\Gamma(b)\Gamma(c-b)} \int_0^1 t^{b-1}(1-t)^{c-b-1}(1-tz)^{-a}dt. $$

De plus si $\Re(b) > \Re(a) > 0$, *on a*

$$ {}_1F_1(a;b;z) = \frac{\Gamma(b)}{\Gamma(a)\Gamma(b-a)} \int_0^1 e^{zt}t^{a-1}(1-t)^{b-a-1}dt. $$

Ces formules peuvent être trouvées respectivement p. 505 et p. 558 du *Handbook of Mathematical Functions* par Abramowitz et Stegun [AS73].

2.6.3 Preuve

La preuve du théorème 2.3 se base sur les résultats de la section 2.5, où on a obtenu le coefficient de $[u^k]$ de la fonction $(1-u)^{-c}(1-zu)^{-b}$.

On extrait le coefficient de $[z^n]$ dans $[u^k]f(z,u)$ par une intégrale de Cauchy prise le long d'un contour de Hankel $\Gamma_{1,2,3,4}$.

$$ [z^n][u^k]f(z,u) = \frac{1}{2i\pi}\int_{\Gamma_{1,2,3}} [u^k]f(z,u)\frac{dz}{z^{n+1}} + \frac{1}{2i\pi}\int_{\Gamma_4}[u^k]f(z,u)\frac{dz}{z^{n+1}} $$

L'intégrale selon $\Gamma_{1,2,3}$ fournit la contribution principale, alors que celle selon Γ_4 apparaîtra comme un terme d'erreur.

Partie principale. Lorsque z est situé le long du contour $\Gamma_{1,2,3}$, le théorème 2.2 s'applique à $[u^k](1-u)^{-c}(1-zu)^{-b}$, soit

$$ [u^k]\frac{1}{(1-u)^c}\frac{1}{(1-z)^a}\frac{1}{(1-zu)^b} \sim \frac{1}{(1-z)^a}\sum_{i\geq 0}\frac{k^{c+b-1}}{\Gamma(c+b+i)}\frac{\Gamma(b+i)}{\Gamma(b)}\frac{(z-1)^i k^i}{i!}. $$

La deuxième partie de la proposition 2.14 s'applique à cette expression, sous l'hypothèse que $\Re(c) > 0$ et $\Re(b) > 0$. On obtient

$$ [u^k]f(z,u) \sim \frac{k^{c+b-1}}{\Gamma(c)\Gamma(b)}\int_0^1 e^{zkt}(1-z)^{-a}e^{-kt}t^{b-1}(1-t)^{c-1}dt. $$

Ceci se récrit

$$ \frac{1}{2i\pi}\int_{\Gamma_{1,2,3}}[u^k]f(z,u)dz \sim \frac{1}{2i\pi}\int_{\Gamma_{1,2,3}}\frac{k^{c+b-1}}{\Gamma(c)\Gamma(b)}\int_0^1 e^{zkt}(1-z)^{-a}e^{-kt}t^{b-1}(1-t)^{c-1}dtdz. \qquad (2.31) $$

On note $P_1 := \frac{1}{2i\pi}\int_{\Gamma_{1,2,3}} e^{zkt}(1-z)^{-a}dz$. Cette intégrale s'évalue par le changement de variable $z = 1 + \frac{v}{n}$:

$$ P_1 = \frac{1}{2i\pi}\int_{\Gamma_{1,2,3}} e^{zkt}(1-z)^{-a}\frac{dz}{z^{n+1}} = \frac{n^{a-1}e^{kt}}{2i\pi}\int_{\Gamma_{1,2,3}} e^{v\left(\frac{kt}{n}-1\right)}(-v)^{-a}dv. $$

L'étape suivante est de faire le changement de variable $v' = -v\left(\frac{kt}{n} - 1\right)$, lequel est légitime car t est inférieur à 1. L'intégrale devient alors :

$$P_1 \sim \frac{(n-kt)^{a-1}e^{kt}}{\Gamma(a)},$$

de par l'application du corollaire 2.1.

L'équation (2.31) se récrit alors

$$\frac{1}{2i\pi}\int_{\Gamma_{1,2,3}} [u^k]f(z,u)dz \sim \frac{k^{c+b-1}}{\Gamma(c)\Gamma(b)}\int_0^1 t^{b-1}(1-t)^{c-1}\frac{(n-kt)^{a-1}}{\Gamma(a)}dt. \tag{2.32}$$

Termes d'erreur. La partie de l'intégrale le long du chemin Γ_4 doit être traitée différemment, car les résultats du théorème 2.2 ne s'appliquent plus. On revient donc à des méthodes de majorations classiques.

$$I_4 := \int_{\Gamma_4}\int_{0+} \frac{1}{(1-u)^c}\frac{1}{(1-z)^a}\frac{1}{(1-zu)^b}\frac{du}{u^{k+1}}\frac{dz}{z^{n+1}}.$$

On choisit comme contour d'intégration pour u le cercle de centre 0 et de rayon $1 - 1/k$. Cette intégrale double est majorée

$$I_4 = \int_{\Gamma_4}\int_{0+} O(k^c k^b n^a)e^{-\log^2 n}.$$

Cette partie fournit donc une contribution négligeable par rapport à la partie principale (2.32) estimée précédemment.

Calcul de l'intégrale (2.32). La première partie de la proposition 2.14 nous permet d'obtenir

$$[z^n u^k]f(z,u) \sim \frac{k^{b+c-1}n^{a-1}}{\Gamma(a)}\frac{1}{\Gamma(b+c)}{}_2F_1\left(1-a,b;b+c;\frac{k}{n}\right),$$

ce qui est exactement l'énoncé du théorème 2.3.

Preuve du théorème 2.4

La preuve du théorème 2.3 peut être reprise intégralement pour obtenir une preuve du théorème 2.4.

L'intuition est que seule la valeur de la fonction au voisinage du point $(1,1)$ conditionne l'asymptotique de ses coefficients, lorsqu'on connaît les singularités. La multiplication par une fonction analytique qui vaut 1 au point $(1,1)$ n'a alors pas d'influence sur le premier terme de l'asymptotique des coefficients.

2.6.4 Ajout de logarithmes

L'ajout de facteurs logarithmiques se fait exactement comme dans le cas univarié. Les preuves étant une reprise complète du cas sans logarithmes, on se contente ici d'énoncer le résultat.

Proposition 2.15 *Soit*

$$f(z,u) = \left(\frac{1}{1-z}\right)^a\left(\frac{1}{1-zu}\right)^b\left(\frac{1}{1-u}\right)^c\log^\alpha\left(\frac{1}{1-z}\right)\log^\beta\left(\frac{1}{1-zu}\right)\log^\gamma\left(\frac{1}{1-u}\right),$$

avec $\Re(a) > 0$, $\Re(b) > 0$ *et* $\Re(c) > 0$, *et* k *et* n *proportionnels avec* $k < n$. *On a*

$$[z^n u^k] f(z,u) \sim \frac{k^{b+c-1} n^{a-1}}{\Gamma(a)} \frac{1}{\Gamma(b+c)} {}_2F_1\left(1-a, b; b+c; \frac{k}{n}\right) \log^{\alpha+\beta+\gamma} n.$$

2.6.5 Quelques cas particuliers

Le théorème 2.3 donne une estimation de l'asymptotique des coefficients de la série génératrice $(1-z)^{-a}(1-zu)^{-b}(1-u)^{-c}$ valable si les parties réelles de a, b et c sont strictement positives. Ce théorème peut être étendu au cas où ces valeurs sont positives ou nulles.

Cas où $c = 0$. Lorsque $c = 0$, on connaît déjà l'asymptotique des coefficients bivariés, obtenue dans la section 2.2.2, à savoir

$$[z^n u^k](1-z)^{-a}(1-zu)^{-b} \sim \frac{k^{b-1}}{\Gamma(b)} \frac{(n-k)^{a-1}}{\Gamma(a)}.$$

En évaluant le résultat du théorème 2.3 en $c = 0$, on obtient

$$[z^n u^k] f(z,u) \sim \frac{k^{b-1} n^{a-1}}{\Gamma(a)} \frac{1}{\Gamma(b)} {}_2F_1(1-a, b; b; \frac{k}{n}) = \frac{k^{b-1} n^{a-1}}{\Gamma(a)\Gamma(b)} \sum_i \frac{\Gamma(1-a+i)}{\Gamma(1-a) i!} (k/n)^i.$$

On reconnaît alors $\sum_i \frac{\Gamma(1-a+i)}{\Gamma(1-a)i!}(k/n)^i = (1-k/n)^{a-1}$, ce qui fait que l'énoncé est établi dans ce cas particulier. Le cas $a = 0$ est analogue.

Cas où $b = 0$. Lorsqu'on spécialise $b = 0$ dans la formule du théorème 2.3, on obtient

$$f_{n,k} \sim \frac{n^{a-1}}{\Gamma(a)} \frac{k^{c-1}}{\Gamma(c)},$$

ce qui est le résultat attendu. Le théorème est donc valable si $b = 0$.

2.7 Fréquence d'apparition de symboles

L'étude de fréquence d'apparition de motifs dans des chaînes de caractères donne parfois lieu à des séries génératrices possédant des singularités confluentes. Pour l'exemple présenté dans cette section, cela conduit à une distribution de fréquence qui est à la limite uniforme sur un domaine.

Le problème étudié dans cette section, directement extrait de [DGL03], concerne la fréquence d'occurrence d'un symbole dans un texte aléatoire. On suppose que ce texte est construit sur un alphabet à deux lettres $\{a, b\}$, et que le symbole recherché est a. Le texte est issu d'une source représentée par un automate. Les transitions de cet automate ont un certain poids, et sont étiquetées par a ou b. Au départ, l'automate est dans l'un de ses états initiaux ; à chaque étape une transition est choisie selon son poids et l'automate change d'état en émettant la lettre correspondant a la transition choisie. La concaténation des lettres émises constitue le texte étudié. Ce modèle de source est formalisé par une matrice $\mu(x)$ de taille $k \times k$, où k est le nombre d'états de l'automate. La valeur $\mu(x)[i,j]$ est le poids de la transition de l'état i vers l'état j etiquetée par x.

2.7.1 Cas primitif

Ce cas de figure est étudié dans [BCGL03], et est rappelé rapidement ici. On note Y_n la variable aléatoire représentant la fréquence d'apparition du symbole a dans un texte aléatoire de taille n produit par la source. On montre que la série génératrice des moments de Y_n est égale à $(Ae^z + B)^n$ à normalisation près, où $A = \mu(a)$ et $B = \mu(b)$. La moyenne et les moments successifs de Y_n sont obtenus en évaluant cette série ou ses dérivées en $z = 0$. Pour obtenir ces moments, on introduit la série génératrice $H(z, w) = \sum (Ae^z + B)^n w^n = [I - w(Ae^z + B)]^{-1}$, et on rappelle la définition des matrices primitives ainsi que quelques unes de leur propriétés liées a la théorie de Perron-Frobenius [Sen81, chap 1].

Définition 2.4 *Une matrice T à coefficients positifs est dite* primitive *s'il existe un entier m tel que tous les coefficients de T^m soient strictement positifs. On dit que l'automate est une représentation primitive, ou que la source est primitive, si la matrice $M = \mu(a) + \mu(b)$ est primitive.*

La primitivité de la représentation assure qu'à partir de chaque état, on peut atteindre tous les autres états.

Lorsque la source qui émet le mot est primitive, on sait par le théorème de Perron-Frobenius que la matrice M possède une unique valeur propre λ de module maximal, de plus que λ est réelle positive. Soit v^T et u les vecteurs propres associés à λ, respectivement à gauche et à droite. Les puissances de M peuvent alors s'écrire comme $M^n = \lambda^n(uv^T + C(n))$. On note $C = \sum_n C(n)$. Ces notations permettent d'énoncer la proposition suivante pour Y_n, qui est le théorème 2 de [BCGL03], et est essentiellement un résultat classique.

Proposition 2.16 *Dans le cas primitif, la moyenne et la variance de Y_n vérifient*

$$\mathbb{E}(Y_n) = \beta n + O(1), \quad \mathbb{V}(Y_n) = \gamma n + O(1),$$

où β et γ sont définis en fonction de λ et des vecteurs propres de M associés à la valeur propre λ.

$$\beta = \frac{v^T A u}{\lambda} \quad \gamma = \beta - \beta^2 + 2\frac{v^T A C A u}{\lambda^2}.$$

De plus, lorsque n tend vers l'infini, Y_n est asymptotiquement gaussienne, c'est-à-dire qu'on a la convergence en loi

$$\frac{Y_n - \beta n}{\sqrt{\gamma n}} \xrightarrow{d} \mathcal{N}(0, 1).$$

2.7.2 Le modèle à deux composantes

Lorsqu'on relâche la condition de primitivité, les phénomènes changent qualitativement de nature. L'article [DGL03] étudie un cas de source non primitive où l'automate qui représente la source est composé de deux parties primitives.

Dans ce cas l'automate qui représente la source est composé de deux parties. Chaque élément est accessible depuis tous les éléments de la même partie que lui, mais entre les deux parties, les déplacements sont à sens unique : il est possible de quitter la première partie pour la deuxième, mais pas de revenir à la première en partant de la deuxième. En d'autre termes, la matrice μ qui code les transitions est alors triangulaire supérieure par blocs, et en s'inspirant des notations du cas primitif, on note

$$A = \mu(a) = \begin{pmatrix} A_1 & A_0 \\ 0 & A_2 \end{pmatrix} \quad B = \mu(b) = \begin{pmatrix} B_1 & B_0 \\ 0 & B_2 \end{pmatrix} \text{ et } M = A + B = \begin{pmatrix} M_1 & M_0 \\ 0 & M_2 \end{pmatrix}.$$

Dans le modèle à deux composantes, on suppose que les matrices M_1 et M_2 sont primitives.

Les résultats de Perron-Frobenius sur les matrices primitives s'appliquent séparément à M_1 et M_2, lesquelles ont chacune une valeur propre réelle dominante, ces valeurs propres étant notées respectivement λ_1 et λ_2. La distribution de la fréquence d'apparition du symbole recherché dépend alors des positions respectives de λ_1 et λ_2. Si l'une des deux valeurs propres est strictement supérieure à l'autre, c'est le cas général, sa composante a tendance à dominer. Si au contraire elles sont égales, c'est le cas dit *équipotent*, et des phénomènes nouveaux apparaissent.

Cas général

Le cas général est résumé rapidement. On suppose qu'il n'y a pas dégénérescence, c'est-à-dire qu'aucune des matrices étudiées n'est nulle. Cette restriction n'est pas présente dans [DGL03].

Lorsque les valeurs propres dominantes associées aux deux composantes sont distinctes (par exemple $\lambda_1 > \lambda_2$), la composante de la plus grande valeur propre a tendance à dominer, et on retrouve les résultats du cas primitif.

Proposition 2.17 *Sous le modèle à deux composantes, si $\lambda_1 > \lambda_2$ on a la convergence en loi*

$$\frac{Y_n - \beta_1 n}{\sqrt{\gamma_1 n}} \to \mathcal{N}(0,1).$$

Les constantes β_1 et γ_1 correspondent aux constantes β et γ du cas primitif, pour la matrice M_1.

Cas équipotent

Dans le cas dit équipotent, les deux composantes apportent leur contribution, et les distributions limites sont alors différentes du cas général. Notons $\lambda_1 = \lambda_2 = \lambda$ la valeur propre commune.

Si on suppose que les constantes des espérances β_1 et β_2 sont différentes, on a

$$H(z,w) = \frac{S(z)w}{(1-y_1(z)w)(1-y_2(z)w)} +) \left(\frac{1}{1-y_1(z)w}\right) + O\left(\frac{R_2(z)}{1-y_2(z)w}\right).$$

On observe un phénomène de confluence de singularités lorsque $w = \lambda^{-1}$ et $z = 0$, car $y_i(0) = \lambda$, $i = 1, 2$.

Les deux singularités étant polaires d'ordre 1, on attend des phénomènes de même nature que pour la série $(1-z)^{-1}(1-zu)^{-1}$, c'est à dire une distribution uniforme sur un certain intervalle. On obtient ici le théorème 15 de [DGL03].

Théorème 2.5 *Si $M_0 \neq 0$, $\lambda_1 = \lambda_2 = \lambda$ et $\beta_1 < \beta_2$, alors la distribution de $\frac{Y_n}{n}$ converge vers la distribution uniforme sur l'intervalle $[\beta_1, \beta_2]$.*

Le cas $\beta_2 < \beta_1$ est similaire

2.8 Quickselect–Quicksort partiel

Les algorithmes de la famille de quicksort fournissent de nombreuses applications du savoir faire d'analyse de singularité. On a vu dans la section 1.3 une analyse d'un quicksort optimisé, dans un cadre d'analyse univariée. La section 2.3 a présenté une étude bivariée de paramètres des arbres binaires de recherche (qui sont liés à des exécutions de quicksort). Cette section fournit une étude de deux algorithmes, quickselect et quicksort partiel, au cours de laquelle apparaissent des séries génératrices du type de (2.28), étudiées dans la section 2.6.

2.8.1 Quickselect

L'algorithme quickselect a été crée par Hoare [Hoa61] à la même époque que le fameux quicksort. Cet algorithme, basé sur le principe de diviser pour régner, permet de sélectionner le m-ième élément dans un tableau de taille n.

L'algorithme

L'algorithme quickselect, qui est très proche de quicksort, est rappelé ici

Quickselect(T,deb,fin,m)
Choisir un pivot p tel que deb≤ p≤fin
Partitionner le tableau
Soit k la position du pivot p
Si k<m **Alors Quickselect**(T,k+1,fin,m-k)
Sinon **Quickselect** (T,deb,k-1,m)

Diverses stratégies de choix du pivot sont classiques, on choisit ici le choix uniforme, implanté par exemple en prenant le premier élément comme pivot. Cet algorithme a en moyenne un coût linéaire, comme on le voit dans la section suivante qui explicite l'analyse en moyenne du coût d'une recherche fructueuse.

L'analyse

On ne présente ici que le cas particulier où le pivot est choisi parmi les éléments avec probabilité uniforme. Ce cas suffit à montrer que des singularités confluentes apparaissent. Des cas plus généraux sont traités dans [MR01, MPV04].

Soit $Q_{n,m}$ le coût moyen de recherche du m-ième élément dans un tableau de taille n. La suite $Q_{n,m}$ vérifie la récurrence

$$Q_{n,m} = n - 1 + \frac{1}{n}\sum_{k=1}^{m-1} Q_{n-k,m-k} + \frac{1}{n}\sum_{k=m+1}^{n} Q_{k-1,m}.$$

Cette récurrence met en évidence la taille des sous-tableaux dans lesquels se font la recherche, celles-ci étant déterminées par la position du pivot. Soit $Q(z,u)$ la série génératrice associée à $Q_{n,m}$, $Q(z,u) = \sum Q_{n,m} z^n u^m$. Cette série génératrice satisfait une équation différentielle issue de la récurrence qui est

$$zQ'_z(z,u) = \frac{2}{(1-z)^3(1-u)} + Q(z,u)\left(\frac{zu}{1-zu} + \frac{z}{1-z}\right).$$

Cette équation différentielle se résout sans difficulté,

$$Q(z,u) = \frac{1}{(1-z)(1-zu)}\left(\frac{2}{1-z} + \frac{2}{1-zu} + \frac{2}{1-u}\left(\log\frac{1}{1-z} - \log\frac{1}{1-zu}\right) + p(z,u)\right),$$

où $p(z,u)$ est un polynôme en z et u. Cela entraîne, d'après les résultats des sections précédentes

$$Q_{n,m} \sim 2(n-m) + 2m + 2m\log(n-m) - 2m\log m \sim 2n.$$

En d'autres termes, le coût moyen de recherche du m-ième élément dans un tableau par l'algorithme quickselect est asymptotiquement équivalent à $2n$, lorsque m et $n-m$ sont proportionnels à n.

Dans ce cas particulier de quickselect à pivot aléatoire uniforme, on voit des termes $\frac{1}{(1-z)(1-zu)}$ apparaître dans la série génératrice. Dans le cas d'une stratégie de choix de pivot plus complexe, cet aspect des séries génératrices est conservé.

L'algorithme *quickselect multiple* généralise l'algorithme quickselect à la recherche de plusieurs éléments. Cet algorithme est étudié par Prodinger dans [Pro96].

L'algorithme quickselect est généralisé de la manière suivante. On recherche simultanément p éléments $j_1 \leq \cdots \leq j_p$. Pour cela on choisit un pivot, et on partitionne le tableau. Ensuite on recherche récursivement les éléments j_k strictement inférieurs au pivot dans le sous-tableau des petits, et les éléments strictement supérieurs au pivot dans le sous-tableau des grands. Si un élément recherché est égal au pivot, alors il est trouvé et sa recherche s'arrête.

Les méthodes de séries génératrice présentées pour l'analyse de l'algorithme quicksect se généralisent sans aucun doute à l'analyse de quickselect multiple, ce qui permettrait de redémontrer les résultats de [Pro96].

2.8.2 Quicksort partiel

Le but de l'algorithme quicksort partiel du à Conrado Martinez [Mar04] est de trier une partie seulement d'un tableau. Partant d'un tableau de n éléments, l'algorithme renvoie les m premiers éléments($m \leq n$) triés.

L'algorithme

L'algorithme quicksort partiel imite l'algorithme quicksort en n'effectuant que les appels utiles, c'est-à-dire que si tous les éléments d'un sous tableau sont situés au delà du m-ième, ce tableau n'est pas trié. L'algorithme complet est décrit dans la figure 2.8.2.0.

```
Quicksort Partiel(T,deb,fin,m)
    Choisir un pivot p tel que deb≤ p≤ fin
    Partitionner le tableau
    Soit k la position du pivot p
    Si k<m Alors Quicksort Partiel(T,deb,k-1,k-1)
                 Quicksort Partiel(T,k+1,fin,m-k)
    Sinon  Quicksort Partiel (T,k+1,fin,m)
```

La méthode de choix de pivot n'est pas précisée a priori, toutes les stratégies classiques peuvent être utilisées.

L'analyse

Les résultats présentés ici sont extraits de l'article Partial Quicksort de Conrado Martinez [Mar04]. Il est plaisant de constater qu'ils s'insèrent naturellement dans le cadre de ce chapitre, ainsi que nous allons l'expliquer brièvement. On suppose, comme il est classique de le faire, que toutes les permutations des n éléments du tableau sont équiprobables.

On note $P_{n,m}$ le nombre moyen de comparaisons effectuées par l'algorithme pour trier les m plus petits éléments d'un tableau de taille n, et $P(z,u)$ sa série génératrice bivariée : $P(z,u) = \sum_n \sum_{m\leq n} P_{n,m} z^n u^m$.

La méthode utilisée pour choisir le pivot n'est pas précisée a priori, et on note $\pi_{n,k}$ la probabilité que le pivot choisi soit le k-ième des n éléments du tableau. Comme on a vu dans la section 1.3, si le choix est uniforme, on a $\pi_{n,k} = n^{-1}$. Si on choisit des stratégies plus évoluées, comme médiane de 3, ou médiane 3-3, la distribution de $\pi_{n,k}$ n'est alors plus uniforme. On suppose dans la suite, pour une présentation simplifiée, que le coût du choix du pivot est une constante c.

La séquence $P_{n,m}$ satisfait la récurrence

$$P_{n,m} = n - 1 + c + \sum_{k=1}^{m} \pi_{n,k}(P_{k-1,k-1} + P_{n-k,m-k}) + \sum_{k=m+1}^{n} \pi_{n,k} P_{k-1,m}.$$

Cette récurrence code le fait que l'algorithme commence par choisir un pivot puis compare tous les éléments à ce pivot (coût $n-1-c$). Ensuite il y a deux situations. Soit le pivot k est inférieur à m, auquel cas il faut trier le premier sous tableau et continuer l'algorithme sur le deuxième sous tableau ($P_{k-1,k-1} + P_{n-k,m-k}$), soit k est supérieur à m, auquel cas on se contente de poursuivre l'exécution de l'algorithme dans le premier sous-tableau ($P_{k-1,m}$).

Lorsque $n = m$, l'algorithme quicksort partiel devient un quicksort total, dont on sait calculer la complexité pour les stratégies de choix de pivot classiques, voir la section 1.3. On note $q_n = P_{n,n}$.

Soit $t_{n,m} = n - 1 + c + \sum_{k=1}^{m} \pi_{n,k} q_{k-1}$, la fonction qui représente la fonction de péage. La récurrence précédente peut se récrire

$$P_{n,m} = t_{n,m} + \sum_{k=1}^{m} \pi_{n,k} P_{n-k,m-k} + \sum_{k=m+1}^{n} \pi_{n,k} P_{k-1,m}.$$

Cette récurrence ressemble beaucoup à celle qu'on observe pour l'algorithme quickselect. Si on note T la série génératrice de $t_{n,m}$, cette récurrence peut être transformée en équation différentielle pour $P(z,u)$, qu'on présente ici sous forme résolue :

$$P(z,u) = \frac{1}{(1-z)(1-zu)} \left(\int (1-z)(1-uz) \frac{\partial T}{\partial z} dz + K \right).$$

On constate alors que si on connaît $T(z)$, ou encore si on dispose d'informations sur le comportement de $\int (1-z)(1-uz)\frac{\partial T}{\partial z} dz$, on peut obtenir des résultats asymptotiques sur $P_{n,m}$, sans avoir besoin de connaître explicitement la série génératrice P. Dans les cas particuliers classiques, où on a accès à la série génératrice, cette théorie bivariée permet de simplifier grandement les calculs, car elle est utilisable directement sur ce type d'exemples.

Pour le cas particulier où $\pi_{n,k} = 1/n$, le coût moyen de l'algorithme quicksort partiel, en terme de nombre de comparaison est trouvé dans [Mar04] et est

$$P_{n,m} = 2n + 2(n+1)H_n - 2(n+3-m)H_{n+1-m} - 6m + 6.$$

Chapitre 3

Hachage avec pagination

Sommaire

Ce chapitre s'intéresse au hachage comme étant un moyen de stocker un grand nombre d'éléments, tout en conservant un accès rapide à chacun d'entre eux. L'algorithme de hachage analysé ici est le *hachage à essai aléatoire*, ou random probing hashing, dans un contexte de pagination. Les résultats présentés ici se basent en partie sur l'article de Larson [Lar83].

Résultats principaux. On obtient dans le Théorème 3.1 la série génératrice multivariée f dont le coefficient $f_{n,k,\underline{\alpha}}$ représente la probabilité qu'une table de hachage qui contient n éléments, ait α_i cases remplies par i éléments et un coût de construction égal à k.

Cette série génératrice permet ensuite d'analyser l'asymptotique du coût de construction. Les théorèmes 3.2 et 3.3 explicitent l'asymptotique de l'espérance et de la variance dans le cas des tables presque pleines ($n = bm - 1$). On obtient que l'espérance est équivalente à $m \log m$ et que l'écart-type est d'ordre $O(m)$.

Pour le cas essentiel des *tables α-pleines*, où le taux de remplissage de chaque case est égal à α, $0 < \alpha < b$, les théorèmes 3.4 et 3.5 montrent que l'espérance du coût de construction est linéaire en m (résultat déjà obtenu par Larson [Lar83]), et que l'écart-type est sous-linéaire en m.

Ces résultats sont issus d'une collaboration avec Alfredo Viola.

Plan. La section 3.1 rappelle les principaux algorithmes de hachage en place, tandis que la section 3.2 présente en détail l'algorithme de hachage à essai aléatoire. Les trois sections suivantes sont consacrées à l'analyse du coût de construction. La partie combinatoire de l'analyse est traitée dans la section 3.3, et les résultats analytiques sont obtenus dans la section 3.4 pour les tables pleines, et dans la section 3.5 pour les tables α-pleines. La dernière section 3.6 constitue une annexe technique.

3.1 Le hachage, utilité et applications

3.1.1 Définitions

On dispose d'un ensemble d'éléments $\mathcal{E} \subseteq \mathcal{U}$, pour un certain univers $\mathcal{U}$ de données, qu'on souhaite stocker dans une table de hachage de taille m. On note $\mathbb{N}_m$ l'ensemble des entiers compris entre 0 et $m-1$.
Une fonction de hachage est une fonction $h : \mathcal{U} \to \mathbb{N}_m$ qui à chaque élément associe un indice de la table de hachage. *Une table de hachage* est un tableau de taille m dont les cases peuvent contenir un élément dans le cas du hachage classique, et plusieurs (mais en nombre toujours inférieur à une borne fixe) dans le cas du hachage avec pagination. La capacité d'une case est notée b. On s'intéresse ici au hachage en place où on se dote d'une stratégie de résolution de collisions. On rappelle qu'à l'opposé, le hachage avec chaînage séparé stocke un nombre arbitraire d'éléments dans chacune de ses cases. Le hachage classique est un cas particulier du hachage avec pagination, avec $b = 1$. Une table est dite *pleine* si elle contient bm éléments,*presque pleine* si elle contient $bm - 1$ éléments et α*-pleine*, si elle contient αm éléments.

L'objectif est de stocker l'ensemble $\mathcal{E}$ dans la table de hachage, et ensuite d'accéder rapidement à chacun des éléments.

On suppose dans la suite que la fonction de hachage est uniforme au sens que toutes les cases sont atteintes avec la même probabilité.

3.1.2 Gestion des collisions

Lorsqu'on tente d'insérer un élément x dans une table de hachage à l'emplacement $h(x)$ indiqué par la fonction de hachage, il arrive que cet emplacement soit déjà rempli. On dit alors qu'il y a une *collision*. Il y a plusieurs manières de résoudre ce problème, et de trouver un endroit où insérer cet élément. Les plus classiques sont présentées ici.

Hachage à essai linéaire. Le *hachage à essai linéaire* (linear probing hashing) est la stratégie de résolution de collision la plus simple. La méthode consiste à insérer l'élément x dans la première case non pleine qui suit $h(x)$. La table est alors considérée comme circulaire, c'est-à-dire que l'emplacement 0 est utilisé après l'emplacement $m-1$. Tant que la table n'est pas pleine, on peut insérer un nouvel élément.

Le défaut de cette méthode est que des îlots d'éléments consécutifs ont tendance a apparaître au fur et à mesure que la table se remplit, et que l'insertion d'un élément peut être coûteuse si sa localisation initiale est au début d'un grand îlot.

Cet algorithme a été très étudié, dans le cas de base par Flajolet, Poblete, Viola et Munro [FPV98, PVM97] et est également connu sous le nom du problème du parking [KW66]. Ce problème peut se formuler comme suit : "une personne conduit sa voiture dans un parking de forme circulaire. Cette personne tourne en rond, et à une position aléatoire sur le parking, décide de se garer. Elle se gare alors dans la première place disponible." On voit que ces deux problèmes sont des formulations différentes de la même question. L'algorithme avec pagination est étudié dans l'article de Viola et Poblete [VP98].

Hachage à essai uniforme. Le *hachage à essai uniforme*, ou uniform probing hashing, résout les collisions en associant à chaque élément, non pas une position dans la table de hachage, mais une permutation de toutes les positions de la table de hachage. Toutes les permutations sont supposées équiprobables. On note ainsi que la fonction de hachage h associe à chaque élément x une

permutation de $\mathbb{N}_m$ notée σ_x. L'élément x tente d'abord de s'insérer à la position $\sigma_x(1)$, si cette case est pleine, alors la deuxième tentative se fait à la position $\sigma_x(2)$, etc.

L'avantage de cette stratégie est qu'elle permet de bien distribuer les éléments dans la table, l'inconvénient est qu'il est difficile et coûteux d'attribuer une permutation aléatoire à chaque élément, et cela même s'il suffit pour la plupart des valeurs de calculer seulement les premiers termes de chaque permutation.

Cet algorithme, ainsi que son analyse, est présenté par Gonnet et Baeza-Yates dans [GBY91], et de façon plus détaillée par Larson dans [Lar83]. L'article de Ramakrishna [Ram88] présente de nombreux résultats expérimentaux.

Hachage à essai double. Le *hachage à essai double* (double hashing) a pour vocation d'imiter le comportement du hachage à essai uniforme, tout en étant plus simple à implanter. La fonction de hachage renvoie une paire d'indices notée $(p_x, s_x$ (p comme position et s comme saut). Les positions successives dans lesquelles x tente de s'insérer sont p_x, $p_x + s_x \mod m$, $p_x + 2s_x \mod m$, etc. A chaque échec on saute de s_x cases et on tente de nouveau d'insérer l'élément.

La valeur p_x a par hypothèse une probabilité uniforme, alors que s_x est choisie de façon uniforme parmi les nombres premiers à m la taille de la table. (Si m est un nombre premier, alors s_x est choisi avec probabilité uniforme entre 1 et $m-1$.) Cette méthode imite le hachage à essai uniforme, car la séquence $p_x + ks_x \mod m$ est une permutation des positions. La différence est que les permutations choisies ne le sont pas uniformément parmi toutes les permutations, puisqu'il s'agit seulement d'un petit sous-ensemble des permutations. On observe de plus que cette stratégie est très simple à implanter, et est peu coûteuse.

Les références pour cet algorithme sont [GBY91, Sed88], ainsi que [GS78, LM93].

Hachage à essai aléatoire. Le *hachage à essai aléatoire* est la méthode de résolution de collision étudiée dans ce chapitre. La fonction de hachage sous-jacente h associe à chaque élément x une séquence de positions qui sont les endroits où l'élément x tente de s'insérer.

Cette méthode ressemble au hachage à essai uniforme, la différence principale étant qu'on autorise des répétitions dans la séquence de positions d'insertions, et qu'un élément peut tenter de s'insérer plusieurs fois dans une même case déjà pleine. Le hachage à essai aléatoire s'avère donc en principe moins bon que le hachage à essai uniforme. Cependant, on lit dans [GBY91] que, pour le cas sans pagination ($b = 1$), lorsque m tend vers l'infini, avec α le taux de remplissage fixé, les performances du hachage à essai aléatoire coïncident au premier ordre asymptotique avec ceux du hachage à essai uniforme.

Les trois méthodes de hachage à essai uniforme, double et aléatoire présentent des caractéristiques asymptotiques comparables dans le cas sans pagination. On peut légitimement espérer que ce soit encore le cas pour les algorithmes avec pagination. Le hachage à essai aléatoire paraît le plus abordable du point de vue de l'analyse, et c'est cette stratégie que nous présentons dans ce chapitre.

Le hachage à essai aléatoire sans pagination est étudié dans [GBY91]. Des résultats concernant le hachage avec pagination sont présentés par Larson [Lar83] et Ramakrishna [Ram89].

3.1.3 Algorithmique

Les algorithmes d'insertion et de suppression présentés dans cette partie sont valables pour toutes les méthodes de résolution de collisions présentées précédemment, et pour des tables avec ou sans pagination.

Insertion - Recherche. Le choix de la méthode de résolution de collisions induit la méthode d'insertion d'un élément. Pour rechercher un élément, il suffit d'imiter le processus d'insertion, en regardant dans l'ordre toutes les cases où l'élément a pu être inséré, ce jusqu'à ce qu'on le trouve, ou bien qu'on arrive sur une case non-pleine (auquel cas la recherche a échoué).

3.1.4 Intérêt de la pagination

De manière simplifiée, un disque magnétique se divise en unité physiques, appelées pages, et accessibles en une seule opération (par positionnement d'une tête de lecture). Ce qui signifie que le coût d'accès à une partie ou à la totalité de la page est le même.

Le hachage avec pagination s'inspire de cela et propose de regrouper les données par page, et de bénéficier de cette organisation en blocs. Le coût d'accès à un bloc de disque mémoire étant largement supérieur au coût d'une comparaison entre éléments de la table, c'est ce paramètre qui est pris en compte dans les estimations de complexité ou de coûts.

3.2 Le hachage à essai aléatoire (Random Probing Hashing)

3.2.1 L'algorithme de hachage aléatoire sans pagination

L'algorithme de hachage aléatoire se base sur une table de hachage de taille m et sur un générateur pseudo-aléatoire g qui prend en argument un élément x qu'on souhaite hacher et un entier positif k, et renvoie en résultat un entier entre 0 et $m-1$. La fonction g fournit donc pour chaque élément à hacher x une séquence de positions possibles, avec d'éventuelles répétitions.

L'algorithme est assez simple. Il insère chaque élément dans la première position libre de sa séquence de positions. L'algorithme s'écrit comme suit en pseudo-code :

Algorithme de hachage aléatoire.
Pour tous les éléments $x \in \mathcal{E}$ **Faire**
  $k := 0$
  Tant que $H[g(x,k)]$ est pleine, **Faire** $k := k+1$
  $H[g(x,k)] := x$

Il est possible qu'un élément échoue plusieurs fois sur la même case pleine. Cela se produit avec une probabilité suffisamment faible pour qu'en pratique ce phénomène soit négligeable.

Une fois la table construite, afin de rechercher un élément, on applique le même type de méthode, c'est-à-dire qu'on teste les positions possibles de l'élément recherché dans l'ordre. En pseudo code cela s'écrit

```
Chercher(x)
  k:=0
  Tant que H[g(x,k)] différent de x et H[g(x,k)] pleine
       Faire k=k+1;
  Si H[g(x,k)]=x  Renvoyer g(x,k)
  Sinon Renvoyer Echec
```

Cette structure de donnée permet de gérer également les suppressions. Les algorithmes de suppression et de recherche peuvent être trouvés dans Knuth [Knu98].

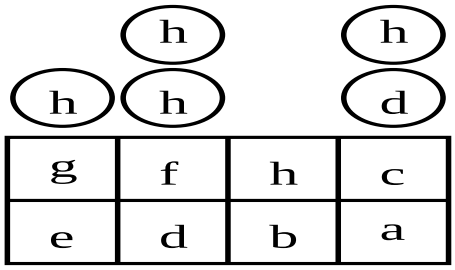

FIG. 3.1 – Un exemple de table de hachage pleine, avec trace des tentatives d'insertion.

3.2.2 Le hachage à essai aléatoire avec pagination

Une table de hachage avec pagination est une table de hachage où chaque case possède une certaine capacité de stockage, généralement notée b. Les paramètres de coût sont exprimés en fonction du nombre de cases manipulées, ce qui généralise le nombre d'éléments lus dans le cas non paginé.

L'algorithme d'insertion étudié dans ce chapitre est le hachage aléatoire avec pagination, qui est une généralisation du hachage aléatoire. Pour l'écriture de l'algorithme en pseudo-code, la table de hachage est notée H et est présentée comme un tableau de dimension 2 de taille $m \times b$.

Algorithme de hachage aléatoire avec pagination.
Initialiser chaque case de $H[][]$ à libre
Pour tous les éléments x à insérer **Faire**
  $k := 0$
  Tant que $H[g(x,k)]$ est plein, **Faire** $k := k + 1$
  $H[g(x,k)][p] := x$, où p désigne le premier emplacement libre de la page.

3.2.3 Exemple

Soit une table de hachage de taille 4, avec la taille des cases b égale à 2. La figure 3.1 montre le remplissage de cette table par 8 éléments numérotés de a à h, et dont les séquences associées sont

a	{4,1,3,2,...}
b	{3,1,3,2,...}
c	{4,1,3,3,...}
d	{4,2,4,2,...}
e	{1,3,2,3,...}
f	{2,2,4,2,...}
g	{1,3,1,2,...}
h	{2,2,1,4,3,...}

Les tentatives d'insertion infructueuses sont représentées sur la figure 3.1 par un cercle contenant l'élément, placé au dessus de la case concernée.

3.2.4 Survol

Per-Åke Larson dans [Lar83] étudie le hachage à essai aléatoire avec pagination dans le cas des tables α-pleines infinies (c'est-à-dire selon un modèle probabiliste correspondant à n et m tendant

vers l'infini). Il obtient l'espérance du coût de construction, sous forme implicite. Ces résultats sont étendus dans la section 3.5. Les méthodes de l'article [Lar83] ne peuvent apparemment pas être étendues au cas où les tables sont finies, et ne permettent pas le calcul des moments d'ordre supérieur.

M. V. Ramakrishna dans [Ram87] étudie le cas particulier des tables où il n'y a pas de débordement, et en calcule la probabilité. Dans l'article [Ram89], il obtient une récurrence qui exprime la distribution du coût de construction. Cette récurrence ne permet pas d'obtenir de résultat explicite ou asymptotique sur le coût de construction, mais permet l'obtention simple de résultats numériques pour des petites valeurs des paramètres.

3.3 Analyse, séries génératrices

L'analyse du hachage aléatoire avec pagination est basée sur une traduction du problème en séries génératrices, puis sur l'étude de ces séries génératrices, grâce en particulier à une variante originale de la méthode de Laplace.

La *répartition* des éléments dans une table est décrite par un vecteur $\underline{\alpha} = (\alpha_1, \ldots, \alpha_b)$, où α_i compte le nombre de cases qui contiennent i éléments. Dans la suite on nomme *répartition* de la table la répartition des éléments. Le *coût de construction* d'une table est égal au nombre de tentatives d'insertion qui ont eu lieu lors de la construction de la table. Cette valeur (aléatoire) est généralement notée k dans ce chapitre.

On considère dans l'analyse que les générateurs de séquences sont aléatoires uniformes. Etant donné un ensemble de n éléments, on considère alors la probabilité $f_{n,k,\underline{\alpha}}$ que le coût d'insertion de ces n éléments soit égal à k et que la répartition de la table soit $\underline{\alpha}$. Pour plus de lisibilité, on note $\underline{x} = x_0, \ldots, x_b$ et $\underline{x}^{\underline{\alpha}} = x_0^{\alpha_0} \ldots x_b^{\alpha_b}$. Le théorème suivant explicite la série génératrice de ces probabilités.

Théorème 3.1 *La série génératrice*

$$f(z,t,\underline{x}) = \sum_{n,k,\underline{\alpha}} f_{n,k,\underline{\alpha}} z^n t^k \underline{x}^{\underline{\alpha}} \tag{3.1}$$

qui code la répartition de la table et le coût de construction est égale à

$$\frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} a(\underline{x},z,s)^{m-1} \left(\sum_{i=0}^{b-1} \frac{(sz/m)^i}{i!} x_i + (1-t)x_b z^b \left(e^{\frac{s}{m}} - \sum_{i=0}^{b-1} \frac{(s/m)^i}{i!} \right) \right) ds, \tag{3.2}$$

où $a(\underline{x},z,s) = \sum_{i=0}^{b-1} \frac{(sz/m)^i}{i!} x_i + x_b z^b \left(e^{\frac{s}{m}} - \sum_{i=0}^{b-1} \frac{(s/m)^i}{i!} \right)$.

Dans la suite, on note le reste de la série exponentielle

$$R_b(z) = e^z - \sum_{i=0}^{b-1} \frac{z^i}{i!}. \tag{3.3}$$

La preuve de ce théorème utilise une modélisation par un problème d'urne, lequel est résolu grâce aux méthodes de [FGP03]. Les résultats obtenus pour les urnes sont ensuite traduits en résultats concernant le hachage. Cette preuve occupe toute la suite de cette section.

3.3.1 Des urnes

L'algorithme de hachage aléatoire avec pagination peut être modélisé par un problème de tirages de boules dans une urne. Cette modélisation permet d'utiliser les méthodes de Flajolet, Gabarró et Pekari [FGP03], et d'obtenir une série génératrice multivariée qui code la répartition de la table et le coût de construction.

La table de hachage contient m cases, chacune de capacité b. Chaque case peut avoir $b+1$ états différents, car elle contient de 0 à b éléments. L'urne qui représente la table contient m boules (chacune représentant une case) qui peuvent être de $b+1$ couleurs différentes. Les couleurs sont notées $0,\ldots,$ b et leur valeur correspond au remplissage de la case qui leur est associée. Au départ, l'urne contient m boules de couleur 0, et lors de chaque insertion dans la table de hachage, il y a une boule qui passe de la couleur i à la couleur $i+1$.

Unité de temps. La quantité de base qu'on souhaite fixer pour étudier le hachage aléatoire avec pagination est le nombre d'insertions. Or vis à vis de la description en urnes, il est plus simple de se baser sur le nombre de tentatives d'insertion (et c'est différent : on peut échouer si on tente d'insérer dans une case pleine) car ainsi, à chaque étape on tire une boule au hasard, et on agit en conséquence. Dans toute la suite, la variable z associée à l'entier n compte le nombre d'insertions, et la variable t associée à l'entier k compte le nombre de tentatives d'insertion.

Une urne est spécifiée par deux données. Tout d'abord un état initial (ici m boules de couleur 0), et ensuite une matrice T de taille $b+1 \times b+1$ qui code les transitions possibles. A chaque étape, une boule est choisie au hasard (toutes les boules sont équiprobables) et est ensuite replacée dans l'urne. De plus si sa couleur est i, on ajoute $T_{i,j}$ boules de couleur j pour toutes les couleurs j.

La matrice suivante correspond au modèle d'urne qui décrit le processus de hachage :

$$\begin{bmatrix} -1 & 1 & 0 & \ldots & 0 \\ 0 & -1 & 1 & 0 & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ 0 & & \ldots & & 0 \end{bmatrix}$$

En effet, lorsqu'on tire au sort une boule de couleur $i < b$, on l'enlève ($T_{i,i} = -1$) et on ajoute une boule de couleur $i+1$. Lorsqu'on tire au hasard une boule pleine, c'est-à-dire lorsqu'on essaye d'insérer un élément dans une case pleine, on échoue et il ne se passe rien.

Cette matrice a la propriété particulière que la somme de chaque ligne est égale à 0, ce qui correspond au fait que quoi qu'il arrive, l'urne contient toujours m boules. Cette propriété permet d'utiliser certains des résultats de [FGP03]. Ceux-ci traitent le cas des urnes à 2 couleurs de boules. L'étude de l'urne correspondant au hachage aléatoire avec pagination peut être conduite grâce à une extension algébrique de [FGP03]. Pour une étude générale des urnes à plus de trois couleurs, on peut se référer à la thèse de Vincent Puyhaubert [Puy04].

La série génératrice exponentielle qui code l'évolution de l'urne associée au hachage est par définition

$$F(t,\underline{x}) = \sum_{k,\underline{\alpha}} a_{k,\underline{\alpha}} \underline{x}^{\underline{\alpha}} \frac{t^k}{k!}. \tag{3.4}$$

Le coefficient $a_{k,\underline{\alpha}}$ est le nombre de configurations qui aboutissent au temps k à la situation où l'urne contient α_i boules de couleur i pour $0 \leq i \leq b$.

Equations différentielles. La série génératrice F vérifie deux équations différentielles. La première exprime les transitions entre couleurs, et la seconde exprime le fait que le nombre de boules reste constant et égal à m.

Lemme 3.1 *La série génératrice F est solution du système de deux équations aux dérivées partielles,*

$$x_1 \frac{\partial F}{\partial x_0} + \cdots + x_b \frac{\partial F}{\partial x_{b-1}} = \frac{\partial F}{\partial t}, \tag{3.5}$$

et

$$x_0 \frac{\partial F}{\partial x_0} + \cdots + x_b \frac{\partial F}{\partial x_b} - mF = 0. \tag{3.6}$$

Preuve. Dériver la série génératrice F par rapport à x_i revient à multiplier chaque terme $t^k \underline{x}^{\underline{\alpha}}$ par α_i, ce qui revient à compter le nombre de possibilité qu'il y a de tirer une boule de couleur i, et ensuite diminuer α_i de 1, ce qui correspond à retirer une boule de couleur i. La multiplication par x_{i+1} code ensuite l'ajout d'une boule de couleur $i+1$. La première équation différentielle peut alors se lire de la façon suivante : le terme de gauche code toutes les évolutions possibles de l'urne en une étape, et le terme de droite code le fait que le temps augmente de 1.

Pour la seconde équation différentielle, la partie $\sum x_i \frac{\partial F}{\partial x_i}$ multiplie chaque sommant de la série génératrice F par le nombre total de boules dans l'urne. Cette quantité étant égale à m, la somme est égale à mF, d'où le résultat. ■

Ce système d'équations différentielles associé aux conditions initiales, code une et une seule solution. En effet, si on note F une solution de ce système, et telle que $[t^0]F = x_0^m$ (les conditions initiales), alors on voit que chaque coefficient $[t^k]F$ est l'état de l'urne au temps k, qui ne dépend que de $t^{k-1}F$, l'état de l'urne au temps précédent. Par récurrence, chaque coefficient est déterminé de façon unique, et donc la solution F est unique.

Solution du système.

Lemme 3.2 *Soit R_b le reste de l'exponentielle définie en* (3.3)*, la série génératrice (définie en* (3.4)*) qui énumère les évolutions de l'urne associée au hachage vaut*

$$F(\underline{x}, t) = \left(x_0 + x_1 t + \ldots \frac{x_{b-1} t^{b-1}}{(b-1)!} + x_b R_b(t) \right)^m .$$

Preuve. La preuve de ce lemme est très simple. Il suffit de montrer que cette série génératrice est l'unique solution du système du lemme 3.1 vérifiant la condition initiale $[t^0]F = x_0^m$. On vérifie tout d'abord que la fonction $x_0 + x_1 t + \ldots \frac{x_{b-1} t^{b-1}}{(b-1)!} + x_b R_b(t)$ vérifie l'équation différentielle (3.5), et ensuite que si f est solution de cette équation différentielle, alors f^m l'est aussi. Cela prouve que la série génératrice F vérifie la première équation différentielle du lemme.

Le fait que F vérifie la deuxième équation différentielle du lemme est une simple vérification. L'évaluation de F en $t = 0$ montre que les conditions initiales sont les bonnes. L'unicité de la solution a été vue auparavant. La preuve du lemme 3.2 est ainsi achevée.

■

Vérifier que la série génératrice F est la solution recherchée est relativement aisé. Cependant, deviner l'expression de cette série génératrice est plus délicat. En l'occurrence, des méthodes basées sur l´article [FGP03] ont d'abord permis de trouver la solution pour le cas $b = 2$ qui est $(x_0 + x_1 t + x_2(e^t - 1 - t))^m$. Ceci nous a naturellement suggéré la solution générale donnée ci-dessus.

3.3.2 Retour au hachage

Le modèle d'urnes exposé à la section précédente est proche du problème de hachage qui nous intéresse, mais est légèrement différent. La série génératrice g égale à

$$g(\underline{x},t) = F(x,t/m) = \left(x_0 + \frac{x_1 t}{m} + \ldots \frac{x_{b-1}t^{b-1}}{m^{b-1}(b-1)!} + x_b R_b(t/m)\right)^m .$$

est la série génératrice dont le coefficient de $t^k \underline{x}^{\underline{\alpha}}$ vaut la probabilité d'aboutir à la répartition $\underline{\alpha}$ après k tentatives d'insertion.

A partir de chaque configuration $\underline{\alpha}$ de la table de hachage, le nombre d'éléments effectivement insérés peut être trouvé sans difficulté, car une case qui est codée par la variable x_i contient i éléments. On opère donc la substitution $x_i \to x_i z^i$ dans la série g pour obtenir la série génératrice g_1 qui possède une variable z supplémentaire codant le nombre d'éléments effectivement insérés dans la table. La série g_1 est égale à

$$g_1(\underline{x},t,z) = \left(x_0 + \frac{x_1 tz}{m} + \ldots \frac{x_{b-1}t^{b-1}z^{b-1}}{m^{b-1}(b-1)!} + x_b z^b R_b(t/m)\right)^m .$$

On voit ici clairement que pour chaque case non pleine, le coût de construction de la case est égal au nombre d'éléments insérés, car il n'y a pas d'échec ; pour les cases pleines, le nombre d'éléments insérés est b, et le coût de construction dû à cette case est supérieur ou égal à b, car des tentatives d'insertion ont pu échouer.

Exponentielle ou ordinaire ? La série génératrice g_1 est exponentielle en la variable t. Ce choix était nécessaire pour pouvoir obtenir la série génératrice via les modèles d'urnes, mais n'est pas adéquat pour la problématique de hachage. En effet pour calculer des paramètres comme le coût de construction, on doit éliminer les coefficients du type $1/k!$.

Pour transformer une série génératrice exponentielle en série génératrice ordinaire, il suffit d'appliquer une transformée de Laplace : $\mathcal{L}(f)(t) = \frac{1}{t}\int_0^\infty e^{-s/t} f(s)ds$. On obtient alors la série génératrice ordinaire g_2 en appliquant la transformée de Laplace à g_1 :

$$g_2(\underline{x},t,z) = \frac{1}{t}\int_0^\infty e^{-s/t}\left(x_0 + \frac{x_1 sz}{m} + \ldots \frac{x_{b-1}s^{b-1}z^{b-1}}{m^{b-1}(b-1)!} + x_b z^b R_b(s/m)\right)^m . \tag{3.7}$$

Ne pas accepter l'échec - Intuition. Le problème de hachage qu'on souhaite formaliser est le suivant : étant donné l'algorithme de hachage aléatoire avec pagination, trouver la série génératrice $f := \sum a_{n,k,\underline{\alpha}} z^n \underline{x}^{\underline{\alpha}} t^k$ telle que $a_{n,k,\underline{\alpha}}$ soit la probabilité d'être dans la configuration $\underline{\alpha}$ avec un coût k *sachant* que n éléments ont été insérés. Par contraste, la série génératrice g_2 code la probabilité d'être dans la configuration $\underline{\alpha}$ avec n éléments insérés *sachant* que k tentatives d'insertion ont été effectuées. Cette deuxième série génératrice code beaucoup plus de situations que la première, car elle prend en compte les cas où on s'arrête sur une tentative d'insertion infructueuse alors que pour la première série génératrice, on ne prend pas ces situations en compte. De plus le fait de pouvoir échouer indéfiniment dans ce deuxième modèle rend la série génératrice g_2 singulière en $z = 1$, alors que f est parfaitement définie.

Un exemple

Pour illustrer cette différence, considérons un exemple simple obtenu en choisissant les paramètres $b = 1$ et $m = 2$, soit une table de hachage qui peut contenir 2 éléments.

Il n'y a qu'un moyen de mettre 0 éléments dans une table, ce qui est codé par x_0^2, également un seul moyen de mettre un élément : une case vide et une case pleine, ce qui s'écrit ztx_0x_1. Lorsqu'on insère deux éléments dans la table, la configuration finale est codée par $z^2x_1^2$, et le nombre de tentatives d'insertion dépend du nombre d'échecs lors de l'insertion du deuxième élément. Il est inséré directement avec probabilité $1/2$, après un échec avec probabilité $1/4$, etc. Cela se traduit en la série génératrice $t^2/2 + t^3/4\ldots$, soit finalement

$$f = x_0^2 + ztx_0x_1 + z^2x_1^2t^2\frac{1}{2-t}.$$

On retrouve que $[z^n]f$ est une série génératrice de probabilité car si on l'évalue en $x_0 = x_1 = t = 1$, on obtient $1 + z + z^2$. Si on dérive f par rapport à t avant de faire cette évaluation, on obtient la série génératrice du coût moyen de construction qui est pour cet exemple $z + 3z^2$. Cela signifie que l'insertion d'un élément se fait en moyenne en une tentative, alors que l'insertion de deux éléments en nécessite en moyenne trois.

En utilisant la formule (3.7) dans ce cas particulier, on obtient

$$g_2(\underline{x}, t, z) = x_0^2 + 2x_0x_1z\frac{t}{2-t} + x_1^2z^2\frac{t^2}{(1-t)(2-t)}.$$

On observe que cette série est bien singulière en $t = 1$. On voit de plus que $[t^k]g_2$ est une série génératrice de probabilité, car $[t^k]g_2(\underline{1}, t, 1) = 1$. Cette série génératrice peut également se calculer de façon directe pour ce cas particulier, en listant toutes les évolutions possibles de la table de hachage.

On vérifie ici que les séries génératrices f et g_2 sont liées entre elles par une simple relation différentielle.

$$g_2 - \frac{t}{2}x_1\frac{\partial g_2}{\partial x_1} = f.$$

Passage du temps t au temps z

Lemme 3.3 *La série génératrice f de l'énoncé du théorème 3.1 codant le coût de construction d'une tables et la série g_2 donnée par* (3.7) *sont reliées par la relation différentielle suivante :*

$$g_2 - \frac{t}{m}x_b\frac{\partial g_2}{\partial x_b} = f.$$

Dans le cas général, la série génératrice g_2 code toutes les situations qui peuvent exister après k tentatives d'insertion, que la dernière tentative soit fructueuse ou non. Pour obtenir la série f qui ne prend en compte que les situations se terminant par une tentative réussie, on calcule la série génératrice qui code toutes les situations ou la dernière tentative d'insertion est un échec, et ensuite on soustrait.

L'opération $\frac{t}{m}x_b\frac{\partial}{\partial x_b}$ revient à ajouter une tentative d'insertion infructueuse à la fin d'une situation quelconque. Car la dérivation par rapport à x_b revient à choisir une case pleine, la multiplication par x_b dit que cette case est laissée intacte, le facteur t ajoute une tentative d'insertion au compteur, et enfin la division par m est la probabilité d'avoir choisi cette case pleine. Enfin la soustraction de $\frac{t}{m}x_b\frac{\partial g_2}{\partial x_b}$ qui code toutes les situations se terminant par un échec à g_2 qui code toutes les situations aboutit à f qui ne prend en compte que les situations qui s'achèvent par un succès.

Preuve du théorème 3.1. La série génératrice recherchée est la fonction f de l'énoncé du théorème 3.1. Un simple calcul à partir du lemme 3.3 et de l'équation (3.7) donne alors

$$\begin{aligned} f &= g_2 - \frac{t}{m} x_b \frac{\partial g_2}{\partial x_b} \\ &= \frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} \left(a(\underline{x}, z, s)^m - \frac{t}{m} x_b \frac{\partial a(\underline{x}, z, s)^m}{\partial x_b} \right) ds \\ &= \frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} a(\underline{x}, z, s)^{m-1} \left(a(\underline{x}, z, s) - t x_b z^b R_b(s/m) \right) ds \\ &= \frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} a(\underline{x}, z, s)^{m-1} \left(\sum_{i=0}^{b-1} \frac{(sz/m)^i}{i!} x_i + x_b z^b (1-t) R_b(s/m) \right) ds. \end{aligned}$$

Ce qui est l'expression recherchée.

3.4 Asymptotique - Tables pleines

Les tables de hachage pleines représentent le cas limite d'utilisation de tables de hachage. On observe que le coût de construction se dégrade fortement lorsque la table est pleine. Ceci provient du fait que l'insertion des derniers éléments est très longue, car les tentatives d'insertion échouent dans presque tous les cas. Il est intéressant d'étudier ce phénomène et de la quantifier précisément.

Les résultats concernant les tables pleines se traduisent immédiatement en résultats pour le problème classique du collectionneur de coupons.

Pour des raisons de convergence de la série génératrice, on s'intéresse ici aux tables presque pleines, c'est-à-dire qui contiennent $mb-1$ éléments, soit une seule position reste inoccupée. Dans ce cas particulier le coefficient de $[z^n]$ s'exprime simplement :

$$[z^{mb-1}]f = \frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} (x_b R_b(s/m))^{m-1} \frac{(s/m)^{b-1} x_{b-1}}{(b-1)!} ((1-t)m+1).$$

La spécialisation $t = 1, x_i = 1$ dans la série $[z^{mb-1}]f$ permet d'écrire

$$\int_0^\infty e^{-s} (R_b(s/m))^{m-1} \frac{(s/m)^{b-1}}{(b-1)!} = 1, \tag{3.8}$$

car $[z^{mb-1}]f$ est une série génératrice de probabilité.

3.4.1 Collectionneur de coupons

Le collectionneur de coupons achète des tablettes de chocolat Poulain, et collectionne les vignettes. On suppose que les vignettes sont équiprobables. Combien de tablettes devra manger en moyenne le collectionneur avant d'avoir un album plein ? Deux albums pleins ? b albums pleins ?

Ce problème est similaire à l'étude du coût de construction d'une table de hachage par l'algorithme de hachage à essai aléatoire. En effet, chaque vignette Poulain est représentée par une urne, et chaque tablette de chocolat achetée est un élément haché dans la table. La table de hachage est pleine lorsque b albums sont complets.

Donald Newman et Lawrence Shepp montrent dans [NS60] que l'espérance du nombres de vignettes nécessaires à la construction de b albums de taille m est $m(\log m + (b-1)\log\log m + C_b + o(1))$ lorsque m tend vers l'infini. Ces résultats sont retrouvés et affinés dans le théorème 3.2.

3.4.2 Coût de construction - Espérance

Théorème 3.2 *La valeur moyenne du coût de construction d'une table presque pleine de taille m, lorsque les pages sont de taille b est de la forme*

$$m\left(\log m + (b-1)\log\log m + \gamma - 1 + \log\frac{1}{(b-1)!} + (b-1)^2\left(\frac{\log\log m}{\log m}\right) + O\left(\frac{1}{\log m}\right)\right).$$

Remarque. Le coût de recherche moyen dans une table presque pleine est par conséquent asymptotiquement égal à $\frac{1}{b}(\log m + (b-1)\log\log m + \gamma - 1 \log\frac{1}{(b-1)!} + o(1))$.

La preuve de ce théorème repose sur une variante de la méthode de Laplace. La méthode de Laplace traditionnelle renormalise la fonction vers la fonction de base e^{-x^2}. Dans le cas présent, la renormalisation naturelle se fait vers la fonction $e^{-x}e^{-e^{-x}}$. La preuve consiste alors à effectuer le changement de variable adéquat pour faire mettre en évidence l'effet de concentration en la variable m (par l'apparition du terme y^m), puis à déterminer les équivalents asymptotiques appropriés pour les termes restants.

La preuve du théorème 3.2 occupe la suite de cette sous-section.

Mise en équation. Le coût de construction s'obtient en dérivant la série génératrice f par rapport à t, puis en évaluant en $t = 1$.

$$\begin{aligned}\mathbb{E}(cc) &:= \frac{\partial[z^{mb-1}]f|_{x_i=1}}{\partial t}|_{t=1} \\ &= (m+1)\int_0^\infty e^{-s}(R_b(s/m))^{m-1}\frac{(s/m)^{b-1}}{(b-1)!} - m\int_0^\infty se^{-s}(R_b(s/m))^{m-1}\frac{(s/m)^{b-1}}{(b-1)!}\end{aligned}$$

La première intégrale a été évaluée à l'équation (3.8) et vaut 1. Le calcul de l'espérance du coût de construction se réduit donc au calcul de la deuxième intégrale. Un premier changement de variable $s' = s/m$ modifie l'expression de l'intégrale

$$\mathbb{E}(cc) - (m+1) = \frac{m^2}{(b-1)!}\int_0^\infty s^b e^{-s}\left(-e^{-s} - se^{-s}\cdots - \frac{s^{b-1}}{(b-1)!} + 1\right)^{m-1} ds$$

On note $P(s) := -e^{-s} - se^{-s}\cdots - \frac{s^{b-1}}{(b-1)!}e^{-s} + 1$, et on remarque que $P'(s) = \frac{s^{b-1}}{(b-1)!}e^{-s}$. Cela permet d'écrire

$$\mathbb{E}(cc) - (m+1) = m^2\int_0^\infty sP'(s)P(s)^{m-1}ds.$$

La fonction P est croissante sur $[0,\infty]$, car sa dérivée est toujours positive, et est à valeur dans l'intervalle $[0,1]$. L'intégrale $\mathbb{E}(cc)$ est transformée grâce au changement de variable $y = P(s)$,

$$\mathbb{E}(cc) - (m+1) = m^2\int_0^1 P^{(-1)}(y)y^{m-1}dy,$$

où $P^{(-1)}$ représente l'inverse vis à vis de la composition de P.

Lorsque m tend vers l'infini, le comportement de l'intégrale est dicté par le comportement de l'intégrande au voisinage de 1, à cause du terme y^{m-1}. On étudie maintenant le terme $P^{(-1)}(y)$ au voisinage de $y = 1$.

Développement de $P^{(-1)}$. La fonction $P^{(-1)}(y)$ est définie implicitement par l'équation

$$y = -e^{-P^{(-1)}(y)} - P^{(-1)}(y)e^{-P^{(-1)}(y)} \cdots - \frac{P^{(-1)}(y)^{b-1}}{(b-1)!}e^{-P^{(-1)}(y)} + 1. \tag{3.9}$$

En s'inspirant de l'article de Corless, Gonnet, Hare et Jeffrey [CGHJ93] sur la fonction de Lambert, on obtient un développement asymptotique de $P^{(-1)}(y)$ en $y = 1$.

Lemme 3.4 *La fonction $P^{(-1)}(y)$ admet le développement suivant au voisinage de $y = 1$:*

$$P^{(-1)}(y) = \log\frac{1}{1-y} + (b-1)\log\log\frac{1}{1-y} + \log\frac{1}{(b-1)!} + (b-1)^2\frac{\log\log\frac{1}{1-y}}{\log\frac{1}{1-y}} + O\left(\frac{1}{\log\frac{1}{1-y}}\right).$$

Preuve. Pour simplifier les notations, on choisit dès maintenant de noter $L := \log\frac{1}{1-y}$ et $LL := \log\log\frac{1}{1-y}$.

Lorsque y est au voisinage de 1, $P^{(-1)}(y)$ est au voisinage de l'infini. On considère donc l'équation simplifiée obtenue à partir de (3.9)

$$1 - y = e^{-P^{(-1)}(y)}\frac{P^{(-1)}(y)^{b-1}}{(b-1)!}. \tag{3.10}$$

On peut montrer formellement que cette simplification d'induit que des termes négligeables dans l'échelle asymptotique du problème.

C'est le terme exponentiel qui donne le comportement dominant de cette dernière équation, de sorte que le premier terme du développement de $P^{(-1)}(y)$ est du type logarithmique. On pose

$$P^{(-1)}(y) = L + A,$$

qu'on substitue dans (3.10). Cela donne $e^A = \frac{(L+A)^{b-1}}{(b-1)!}$. En supposant $A = o(L)$, cela se réduit à $A \sim (b-1)LL$.

On itère le processus : on pose

$$P^{(-1)}(y) = L + (b-1)LL + B,$$

qu'on injecte dans l'équation (3.10). Cela donne $B \sim -\log((b-1)!)$. L'étape suivante est

$$P^{(-1)}(y) = L + (b-1)LL - \log((b-1)!) + C,$$

qui donne $e^C = \left(1 + (b-1)\frac{LL}{L}\right)$, dont on approche la solution par $C \sim (b-1)^2\frac{LL}{L}$. La dernière étape est de poser

$$P^{(-1)}(y) = L + (b-1)LL - \log((b-1)!) + (b-1)^2\frac{LL}{L} + D,$$

soit $D \sim (b-1)\frac{-\log(b-1!)}{L}$.

D'après l'article [CGHJ93] portant sur le développement de la fonction W de Lambert, on peut montrer que $P^{(-1)}(y)$ admet un développement asymptotique complet mettant en jeu L et LL. On se contentera ici des premiers termes,

$$P^{(-1)}(y) = L + (b-1)LL - \log(b-1!) + (b-1)^2\frac{LL}{L} + O\left(\frac{1}{L}\right),$$

lesquels suffisent à terminer la preuve du lemme 3.4.

■

Le développement obtenu pour $P^{(-1)}$ s'intègre terme à terme, y compris le terme d'erreur, ce qui s'écrit

$$\begin{aligned}\mathbb{E}(cc) - (m+1) &= m^2 \left(\int_0^1 L y^{m-1} dy + (b-1) \int_0^1 LL y^{m-1} dy + \log \frac{1}{(b-1)!} \int_0^1 y^{m-1} dy + \right. \\ &\quad \left. (b-1)^2 \int_0^1 \frac{LL}{L} y^{m-1} dy + O\left(\int_0^1 \frac{1}{L} y^{m-1} dy \right) \right).\end{aligned}$$

Le calcul de chacune de ces intégrales est décrit dans la section 3.6 et ne pose pas de difficulté particulière. On obtient alors le résultat annoncé dans le théorème 3.2

3.4.3 Coût de construction - Variance

Théorème 3.3 *La variance du coût de construction d'une table presque pleine de taille m et dont les pages ont taille b vérifie*

$$\mathbb{V}(cc) = O(m^2).$$

Pour calculer la variance, on commence par étudier le moment d'ordre 2, noté M_2. La dérivée seconde de $[z^{mb-1}]f$ évaluée en $t = 1$ permet d'écrire

$$M_2 + 2m\mathbb{E}(cc) = m^2 \int_0^1 P^{(-1)}(y)^2 y^{m-1} ds + o(m^2). \tag{3.11}$$

Pour calculer l'intégrale du membre droit de cette équation, on reprend le développement de $P^{(-1)}(y)$ trouvé dans la section espérance, et qu'on rappelle ici

$$P^{(-1)}(y) = L + (b-1)LL - \log(b-1!) + (b-1)^2 \frac{LL}{L} + O\left(\frac{1}{L}\right).$$

Le développement de $P^{(-1)}(y)^2$ est alors égal à

$$P^{(-1)}(y)^2 = L^2 + 2(b-1)L.LL - 2L\log(b-1!) + (b-1)^2 LL^2 + LL\big(-2(b-1)\log(b-1!) + 2(b-1)^2\big) + O(1).$$

Ce développement peut être substitué dans l'intégrale de l'équation (3.11), laquelle s'intègre ensuite terme à terme. Le détail du calcul des intégrales est reporté à la section 3.6. Finalement, le moment d'ordre 2 admet le développement asymptotique

$$\begin{aligned}M_2 + 2m\mathbb{E}(cc) &= m^2 \Big(\log^2 m + 2\gamma \log m + 2(b-1)(\log m \log\log m + \gamma \log\log m) \\ &\quad -2\log(b-1!)(\log m) + (-2(b-1)\log(b-1!) + 2(b-1)^2)\log\log m + O(1) \Big).\end{aligned}$$

La variance est égale à $M_2 - \mathbb{E}(cc)^2$, et on constate que tous les termes dominants s'annulent jusqu'à l'ordre m^2, ce qui permet d'écrire

$$\mathbb{V}(cc) = O(m^2).$$

3.5 Tables α-pleines

Les résultats de la section précédente ont montré la dégradation de performance associée à des tables presque pleines. Pour cette raison, il est souhaitable que les tables de hachage manipulées soient au plus α-pleines ($\alpha < b$).

Le coût de construction des tables α-pleines est analysé dans cette section. On retrouve tout d'abord des résultats de Larson [Lar83] en montrant que l'espérance du coût de construction est asymptotiquement équivalent à ρm, où ρ est défini par une équation implicite, et ne dépend que de α. Puis on donne une borne supérieure pour la variance.

3.5.1 Loi de probabilité

La série génératrice qui code l'état d'une table de hachage de taille m a été énoncée dans le théorème 3.1.

$$f(z,t) = \frac{1}{t}\int_0^\infty e^{-\frac{s}{t}} a(\underline{x},z,s)^{m-1}\left(\sum_{i=0}^{b-1}\frac{(sz/m)^i}{i!}x_i + (1-t)x_b z^b R_b(s/m)\right) ds,$$

où $a(\underline{x},z,s) = \sum_{i=0}^{b-1}\frac{(sz/m)^i}{i!}x_i + x_b z^b R_b(s/m)$.

La série génératrice de probabilité du coût de construction notée $C(t)$ s'obtient par spécification de $\underline{x}$ en $\underline{1}$, et en extrayant le coefficient de z^n. On obtient alors

$$\begin{aligned} C(t) &= \sum_{i=0}^{b-1}\frac{m}{i!}\sum_{\substack{\sum_j \beta_j = m-1\\ \sum_j j\beta_j = n-i}}\binom{m-1}{\beta_0,\ldots,\beta_b}\frac{1}{\prod_{j<b} j!^{\beta_j}}\int_0^\infty e^{-\frac{mv}{t}} v^{n-b\beta_b} R_b(v)^{\beta_b} dv \\ &\quad +(1-t)\sum_{\substack{\sum_j \beta_j = m-1\\ \sum_j j\beta_j = n-b}}\binom{m-1}{\beta_0,\ldots,\beta_b}\frac{1}{\prod_{j<b} j!^{\beta_j}}\int_0^\infty e^{-\frac{mv}{t}} v^{n-b\beta_b-b} R_b(v)^{\beta_b+1} dv \end{aligned} \tag{3.12}$$

De par la combinatoire du problème, la série génératrice $C(t)$ évaluée en $t=1$ est égale à 1. Cette évaluation, simple à obtenir, est utile pour calculer les moments suivants du coût de construction paramètre. On l'écrit donc de manière complète :

$$1 = \sum_{i=0}^{b-1}\frac{m}{i!}\sum_{\substack{\sum_j \beta_j = m-1\\ \sum_j j\beta_j = n-i}}\binom{m-1}{\beta_0,\ldots,\beta_b}\frac{1}{\prod_{j<b} j!^{\beta_j}}\int_0^\infty e^{-mv} v^{n-b\beta_b} R_b(v)^{\beta_b} dv \tag{3.13}$$

On détermine maintenant la forme asymptotique des deux premiers moments du coût de construction.

3.5.2 Coût de construction, espérance

L'espérance du coût de construction a été obtenue par Larson [Lar83]. La méthode utilisée ne peut être étendue aux moments suivants. On retrouve ici ce résultat, mais par une méthode qui peur ensuite être étendue au moment d'ordre 2.

Théorème 3.4 *[Larson] L'espérance du coût de construction est asymptotiquement équivalente à $m\rho$ lorsque m tend vers l'infini, où ρ est défini par l'équation implicite*

$$1 = \frac{n - b\beta}{\rho m} + \frac{\beta}{m}\frac{R_{b-1}(\rho)}{R_b(\rho)}, \quad avec \quad \beta = me^{-\rho}\sum_{i=b}^{\infty}\frac{\rho^i}{i!}.$$

La preuve de ce théorème se fait en comparant l'expression de l'espérance à l'équation (3.13), et est principalement basée sur la méthode de Laplace.

L'espérance du coût de construction notée $\mathbb{E}(cc)$ s'obtient en dérivant la série génératrice C par rapport à t, puis en évaluant la série obtenue en $t = 1$.

$$\mathbb{E}(cc) = C'(1). \tag{3.14}$$

Pour la commodité des calculs, on choisit de décaler la série génératrice d'un facteur t, ce qui revient à décaler le coût de construction de 1.

$$\begin{aligned}
\mathbb{E}(cc) &= \sum_{i=0}^{b-1}\frac{m}{i!}\sum_{\substack{\sum_j \beta_j = m-1 \\ \sum_j j\beta_j = n-i}}\binom{m-1}{\beta_0,\ldots,\beta_b}\frac{1}{\prod_{j<b} j!^{\beta_j}}\int_0^\infty \mathbf{mv}e^{-mv}v^{n-b\beta_b}R_b(v)^{\beta_b}dv \qquad (3.15)\\
&\quad - \sum_{\substack{\sum_j \beta_j = m-1 \\ \sum_j j\beta_j = n-b}}\binom{m-1}{\beta_0,\ldots,\beta_b}\frac{1}{\prod_{j<b} j!^{\beta_j}}\int_0^\infty e^{-mv}v^{n-b-b\beta_b}R_b(v)^{\beta_b+1}dv\\
&=: I - J.
\end{aligned}$$

L'étude de ces sommes multiples est a priori complexe. De fait, pour obtenir un équivalent asymptotique de l'espérance du coût de construction, on utilise l'équation (3.13).

Nombre de pages pleines. On constate que les intégrales présentes dans l'expression de $\mathbb{E}(cc)$ ne dépendent que de m, de b et de β_b, ce dernier comptant le nombre de pages pleines dans la table de hachage. Larson, dans l'article [Lar83], a montré que la probabilité que k éléments aient tenté de s'insérer dans une case donnée suit une loi de Poisson de paramètre x le coût de construction moyen de chaque case, qui vérifie l'équation implicite

$$x = \frac{n}{m} - be^{-x}R_b(x) + xe^{-x}R_{b-1}(x).$$

Une page est pleine s'il y a au moins b éléments qui ont tenté de s'y insérer, ce qui a une probabilité $p_f = \sum_{k\geq b} e^{-x}\frac{x^k}{k!}$. Elle est non pleine avec la probabilité complémentaire. Le nombre de pages pleines est donc concentré autour de la valeur moyenne $\beta = mp_f$, qui s'écrit

$$\beta = me^{-x}R_b(x). \tag{3.16}$$

Le coût de construction d'une table de hachage particulière est a priori linéaire, et varie peu selon la répartition des éléments dans la table, comparé à la variation des probabilités des différentes répartitions. On peut donc approcher $\beta_b \sim \beta$ dans les intégrales présentes dans l'expression du coût de construction moyen $\mathbb{E}(cc)$, ou dans l'équation (3.13).

Etude de l'intégrale, pour $m \to \infty$. L'intégrale $\int_0^\infty e^{-mv} v^{n-b\beta_b} R_b(v)^{\beta_b} dv$ avec éventuellement quelques variations mineures intervient dans toutes les expressions ci-dessus.

On a vu précédemment qu'on pouvait étudier uniquement le cas $\beta_b = \beta$ (grâce à la concentration de distribution de β_b), ce qui simplifie beaucoup la situation. On pose donc

$$A = \int_0^\infty e^{-mv} v^{n-b\beta} R_b(v)^\beta dv,$$

qu'on va étudier en appliquant la méthode de Laplace.

On pose $h(v) := -v + \frac{n-b\beta}{m} \log v + \frac{\beta}{m} \log R_b(v)$. Cette fonction ne dépend que du taux de remplissage α de la table et non pas de m. L'intégrale A se récrit sous la forme $\int_0^\infty e^{mh(v)} dv$. En suivant toujours la méthode de Laplace, on observe que la fonction h ne présente qu'un seul maximum sur $[0, \infty]$, et on pose $h'(\rho) = 0$. La valeur ρ est alors définie par l'équation implicite

$$1 = \frac{n - b\beta}{\rho m} + \frac{\beta}{m} \frac{R_{b-1}(\rho)}{R_b(\rho)}.$$

Comme h ne dépend pas de m, la position du maximum n'en dépend pas non plus. La contribution principale de l'intégrale a lieu au voisinage de ce point ρ.

Ce qui distingue l'équation (3.13) et la somme I de l'équation (3.15) est un facteur mv, noté en gras dans l'équation (3.15). L'importance de ce facteur est montrée le lemme suivant.

Lemme 3.5 *L'intégrale $\int_0^\infty v^k e^{mh(v)} dv$ est asymptotiquement équivalente à $\rho^k \int_0^\infty e^{mh(v)} dv$, où ρ est défini par $h'(\rho) = 0$.*

Preuve. L'intégrale $\int_0^\infty v^k e^{mh(v)} dv$ est découpée en trois parties pour extraire le comportement central : $I_1 + I_2 + I_3 := \int_0^\infty = \int_0^{\rho-\epsilon} + \int_{\rho-\epsilon}^{\rho+\epsilon} + \int_{\rho+\epsilon}^\infty$, où ϵ est fixe (et petit). Soit λ un majorant de h sur $[0, \rho - \epsilon] \cup [\rho + \epsilon, \infty[$. La valeur λ est strictement inférieure à $h(\rho)$, de par sa définition. La décroissance de $h(v)$ lorsque v s'éloigne de ρ garantit une décroissance exponentielle du terme à l'intérieur des deux intégrales I_1 et I_3. Leur contribution a alors un comportement asymptotique d'ordre de grandeur exponentiel égal à $e^{m\lambda}$. L'intégrale I_2 se comporte asymptotiquement comme $e^{mh(\rho)}$.

Lorsque m tend vers l'infini, comme $\lambda < h(\rho)$, les intégrales I_1 et I_3 sont négligeables devant I_2. On peut alors écrire qu'asymptotiquement

$$\int_0^\infty v^k e^{mh(v)} dv \sim \int_{\rho-\epsilon}^{\rho+\epsilon} v^k e^{mh(v)} dv. \tag{3.17}$$

La valeur de v dans l'intégrale de droite est encadrée par $\rho - \epsilon$ et $\rho + \epsilon$. Cela permet d'écrire

$$\int_{\rho-\epsilon}^{\rho+\epsilon} (\rho - \epsilon)^k e^{mh(v)} dv \leq \int_{\rho-\epsilon}^{\rho+\epsilon} v^k e^{mh(v)} dv \leq \int_{\rho-\epsilon}^{\rho+\epsilon} (\rho + \epsilon)^k e^{mh(v)} dv.$$

L'équation (3.17) est valable pour $k = 0$, ce qui permet de récrire l'encadrement précédent

$$(\rho - \epsilon)^k \int_0^\infty e^{mh(v)} dv \leq_\sim \int_0^\infty v^k e^{mh(v)} dv \leq_\sim (\rho + \epsilon)^k \int_0^\infty e^{mh(v)} dv. \tag{3.18}$$

Le signe $\leq_\sim$ signifie que les inégalités sont valides asymptotiquement.

Comme les inégalités (3.18) valent quel que soit $\epsilon > 0$, le lemme 3.5 est prouvé.

■

Le cas particulier $k = 1$ du lemme précédent entraîne $I \sim m\rho$.

Bilan. La somme J, définie dans l'équation (3.15) est asymptotiquement négligeable devant I, car elle est égale à $O(1)$.

L'espérance du coût de construction est donc asymptotiquement équivalent à $m\rho$. La quantité ρ est donc le coût de construction moyen d'une case, soit en reprenant les notations de Larson $\rho = x$. L'équation (3.16) donne alors l'expression de β recherchée.

Cela achève la preuve du théorème 3.4.

3.5.3 Coût de construction, variance

Théorème 3.5 *Le moment d'ordre 2 du coût de construction d'une table α-pleine est équivalent à $m^2\rho^2$, ce qui fait que la variance du coût de construction notée $\mathbb{V}$ vérifie $\mathbb{V} = o(m^2)$. Par conséquent la distribution du coût de construction est concentrée.*

Le moment d'ordre 2 du coût de construction est noté M_2. Comme l'espérance, on peut l'obtenir en dérivant la série génératrice $C(t)$ par rapport à t, mais deux fois, puis en l'évaluant en $t = 1$. On obtient alors

$$\begin{aligned} M_2 &= \mathbb{E}(cc) \sum_{i=0}^{b-1} \frac{m}{i!} \sum_{\substack{\sum_i \beta_i = m-1 \\ \sum_i i\beta_i = n-i}} \binom{m-1}{\beta_0, \ldots, \beta_b} \frac{1}{\prod_i i!^{\beta_i}} \int_0^\infty (\mathbf{m^2v^2 - 2mv}) e^{-mv} v^{n-b\beta_b} R_b(v)^{\beta_b} dv \\ &\quad - \sum_{\substack{\sum_i \beta_i = m-1 \\ \sum_i i\beta_i = n-i}} \binom{m-1}{\beta_0, \ldots, \beta_b} \frac{1}{\prod_i i!^{\beta_i}} \int_0^\infty \mathbf{2mv} e^{-mv} v^{n-b-b\beta_b} R_b(v)^{\beta_b+1} dv. \end{aligned}$$

Le terme dominant dans cette expression est celui qui contient le facteur m^2. Ce terme peut être estimé en utilisant les mêmes méthodes que pour l'espérance. Le lemme 3.5 nous dit que

$$\int_0^\infty v^2 e^{mh(v)} dv \sim \rho^2 \int_0^\infty e^{mh(v)} dv,$$

où ρ est défini par $h'(\rho) = 0$.

Cela entraîne que $M_2 \sim m^2\rho^2$, et donc qu'il y a une annulation des deux premiers termes dans le calcul de la variance, ce qui termine la preuve du théorème.

3.5.4 Extension

Cette approche par série génératrice multivariée est à mon sens très riche. On a obtenu ici un équivalent de l'espérance et une borne supérieure de la variance pour le coût de construction. Il est certainement possible d'obtenir bien plus. On voit déjà que l'obtention du premier terme de tous les moments du coût de construction peut être faite sans difficultés. De plus, je pense qu'une analyse plus poussée donnera accès à plus de termes du développement asymptotique de chaque moment du coût de construction. Cette approche permet également d'obtenir des informations sur la répartition des tables.

3.6 Appendice : Estimations d'intégrales

Ce lemme énonce les égalités nécessaires au calcul de l'espérance et de la variance du coût de construction des tables presque pleines. On rappelle que $L = \log \frac{1}{1-z}$ et $LL = \log\log \frac{1}{1-z}$.

Lemme 3.6 *On dispose des évaluations asymptotiques suivantes*

$$m\int_0^1 Ly^{m-1}dy = \log m + \gamma + O\left(\frac{1}{\log m}\right)$$
$$m\int_0^1 LLy^{m-1}dy = \log\log m + O\left(\frac{1}{\log m}\right)$$
$$m\int_0^1 L/LLy^{m-1}dy = \frac{\log\log m}{\log m} + O\left(\frac{1}{\log m}\right)$$
$$m\int_0^1 1/Ly^{m-1}dy = O\left(\frac{1}{\log m}\right)$$
$$m\int_0^1 L^2y^{m-1}dy = \log^2 m + 2\gamma\log m + O(1)$$
$$m\int_0^1 L.LLy^{m-1}dy = \log m\log\log m + \gamma\log\log m + O(1)$$
$$m\int_0^1 LL^2y^{m-1}dy = (\log\log m)^2 + O(1)$$

Le reste de la section est consacré à la preuve de ce lemme. Dans la preuve, il est commode de substituer $m-1 \to m$ dans les intégrales, ce qui n'affecte pas les résultats énoncés.

On commence par montrer le schéma qui est utilisé pour l'estimation de toutes les intégrales du lemme. Soit $f(y)$ une fonction à variation au plus logarithmique. Ici $f(y)$ est égale à L, LL ou une fraction rationnelle en ces deux fonctions. On pose $y = 1 - \frac{a}{m}$

$$\int_0^1 f(y)y^m dy = \frac{1}{m}\int_0^m f\left(1-\frac{a}{m}\right)\left(1-\frac{a}{m}\right)^m da.$$

Cette intégrale est coupée en deux morceaux, par $x_m = m^{1/3}$. La partie $a > x_m$ est majorée par

$$\frac{1}{m}\int_{x_m}^m M_f\left(1-\frac{a}{m}\right)^m da = M_f\left(1-\frac{x_m}{m}\right)^m \sim M_f e^{-x_m},$$

où M_f est un majorant de f sur l'intervalle $[x_m, m]$. Si $f(y) = L$, $M_f = \log m$, si $f(y) = LL$, $M_f = \log\log m$. Dans tous les cas rencontrés dans cette section, la contribution de l'intégrale pour $a > x_m$ est exponentiellement négligeable. Pour la partie de l'intégrale $a < x_m$, on a à notre disposition l'approximation $\left(1-\frac{a}{m}\right)^m = e^{-a}(1+O(x_m^2/m))$.

On remarque de plus

$$\int_0^1 y^m dy = \int_0^m \left(1-\frac{a}{m}\right)^m da = \frac{1}{m+1}$$

Le terme d'erreur $O\left(\frac{1}{\log m}\right)$ recherché pour les premières intégrales est noté $O(l^{-1})$.

– $\boxed{\text{L}}$

$$m\int_0^1 Ly^m dy = \int_0^{x_m}\log(m/a)\left(1-\frac{a}{m}\right)^m da + O(l^{-1}) = \log m - \int_0^{x_m}\log ae^{-a}da + O(l^{-1}).$$

L'intégrale de $\log ae^{-a}$ s'exprime de façon explicite, et on obtient

$$m\int_0^1 Ly^m dy = \log m + \gamma + O(l^{-1}).$$

– LL

$$\begin{aligned} m\int_0^1 LLy^m dy &= \int_0^{x_m} \log\log(m/a)e^{-a}da + O(l^{-1}) \\ &= \log\log m + \int_0^{x_m} \log\left(1-\frac{\log a}{\log m}\right)e^{-a}da + O(l^{-1}). \end{aligned}$$

L'intégrale de l'équation précédente est divisée en deux parties, autour de la valeurs $a = 1$ et on montre

$$\int_0^1 \log\left(1-\frac{\log a}{\log m}\right)e^{-a}da = O(l^{-1}) \quad et \quad \int_1^{x_m} \log\left(1-\frac{\log a}{\log m}\right)e^{-a}da = O(l^{-1}),$$

respectivement par la majoration $\log\left(1-\frac{\log a}{\log m}\right) \leq \frac{\log a}{\log m}$ et la majoration $\left|\log\left(1-\frac{\log a}{\log m}\right)\right| \leq \frac{2\log a}{\log m}$, toutes deux valables sur les intervalles d'intégration concernés.
Finalement on a prouvé

$$m\int_0^1 LLy^m dy = \log\log m + O(l^{-1})$$

– L/LL

$$m\int_0^1 L/LLy^m dy = \int_0^{x_m} \frac{\log\log m/a}{\log m/a}e^{-a}da + O(l^{-1})$$

Les logarithmes se développent, ce qui donne

$$\frac{\log\log m/a}{\log m/a} = \frac{\log\log m}{\log m}\left(\frac{1+\frac{log(1-x)}{\log\log m}}{1-x}\right),$$

où $x = \frac{\log a}{\log m}$. On distingue deux intervalles pour a, le premier est $a \in [0, x_m^{-1}]$, le deuxième est $a \in [x_m^{-1}, x_m]$. Dans le premier intervalle, on borne la fonction L/LL par un majorant M, et dans le deuxième, on peut développer cette fonction en séries, et écrire

$$\frac{\log\log m/a}{\log m/a} = \frac{\log\log m}{\log m}\left(1+O\left(\frac{\log a}{\log m}\right)\right).$$

L'intégration de cet équivalent et de cette majoration donne le résultat recherché :

$$m\int_0^1 L/LLy^m dy = \frac{\log\log m}{\log m} + O(l^{-1})$$

– 1/L

$$m\int_0^1 1/Ly^m dy \int_0^{x_m} \frac{1}{\log m/a}e^{-a}da + O(l^{-1})$$

On utilise les mêmes approximations qu'au point précédent, et le résultat est alors immédiat.

La précision recherchée est maintenant $O(1)$ pour tous les termes qui suivent.

– $\boxed{L^2}$

$$m\int_0^1 L^2 y^m dy = \int_0^{x_m} (\log m - \log a)^2 e^{-a} da + O(1)$$

Le terme $\int_0^{x_m} \log^2 a e^{-a} da$ est en $O(1)$, le reste se calcule d'après les résultats précédents. On obtient alors

$$m\int_0^1 L^2 y^m dy = \log^2 m + 2\log m\gamma + O(1).$$

– $\boxed{L.LL}$

$$m\int_0^1 L.LLy^m dy = \int_0^{x_m} (\log m - \log a)\left(\log\log m + \log\left(1 - \frac{\log a}{\log m}\right)\right)e^{-a} da + O(1)$$

Lorsqu'on développe le produit dans l'intégrale, tous les termes s'intègrent grâce aux résultats précédents, sauf le terme $\log a \log\left(1 - \frac{\log a}{\log m}\right)$. On reprend les majorations écrites lors de l'analyse du terme $\boxed{\text{LL}}$, et on obtient que

$$\int_0^{x_m} \log a \log\left(1 - \frac{\log a}{\log m}\right) e^{-a} da = O(1).$$

Cela montre

$$m\int_0^1 L.LLy^m dy = \log m \log\log m + \gamma \log\log m + O(1).$$

– $\boxed{LL^2}$

$$m\int_0^1 LL^2 y^m dy = \int_0^{x_m} (\log\log m + \log\left(1 - \frac{\log a}{\log m}\right)^2 e^{-a} da + O(1).$$

Lorsqu'on développe le produit dans l'intégrale, tous les termes s'intègrent grâce aux résultats précédents, sauf le terme $(\log\left(1 - \frac{\log a}{\log m}\right)^2$. On reprend les majorations écrites lors de l'analyse du terme $\boxed{\text{LL}}$, et on obtient que

$$\int_0^{x_m} (\log\left(1 - \frac{\log a}{\log m}\right)^2 e^{-a} da = O(1).$$

Cela montre

$$m\int_0^1 LL^2 y^m dy = (\log\log m)^2 + O(1).$$

Chapitre 4

Comptage Probabiliste

Sommaire

Le comptage probabiliste vise à extraire une certaine information qualitative d'un multiensemble d'éléments, potentiellement très grand. L'information recherchée est une estimation du nombre d'éléments distincts, encore appelé cardinalité dans l'ensemble étudié, ce rapidement et avec peu de mémoire. Cette question fait partie d'une problématique plus générale qui est de caractériser au moins partiellement la distribution de la fréquence des éléments dans un ensemble de données, dont l'origine et la distribution sont a priori inconnues. En reprenant les notations de Alon, Matias et Szegedy [AMS99], on note $F_p = \sum x_i^p$, où x_i est le nombre de d'occurrences de l'élément i ; ce chapitre est alors consacré à la détermination approchée de la quantité F_0 qui représente la cardinalité. Comme les informations visées incluent des flots massifs de données, la difficulté du problème provient du fait que la mémoire disponible est extrêmement petite, et ne permet absolument pas de stocker ne serait-ce qu'une fraction non négligeable des éléments. L'objectif est d'obtenir la meilleure estimation possible, tout en minimisant la quantité de mémoire utilisée.

Résultats principaux. Le théorème 4.1 énonce les résultats concernant l'analyse de l'algorithme Loglog. Lorsque l'algorithme *Loglog* est appliqué à un multiensemble idéal de cardinalité n, et si on note E_n la valeur retournée par l'algorithme alors

(i) L'estimateur E_n est asymptotiquement non biaisé, c'est-à-dire que, lorsque $n \to \infty$, la quantité $\frac{1}{n}\mathbb{E}(E_n)$ est voisine de 1 (à des fluctuations d'amplitude très faible près).

(ii) Lorsque m mots mémoire sont utilisés, l'erreur standard, mesurée comme $\frac{1}{n}\sqrt{\mathbb{V}(E_n)}$ vérifie quand $n \to \infty$, $\frac{1}{n}\sqrt{\mathbb{V}(E_n)} \sim \frac{\beta_m}{\sqrt{m}}$ (également à des fluctuations d'amplitude très faible près), où les valeurs de β_m sont proches de 1.3, et sont décrites précisément dans le théorème 4.1.

Les propositions 4.4 et 4.5 donnent l'espérance et la variance d'un algorithme de comptage par collisions utilisé comme procédure de correction au chapitre suivant.

4.1 Présentation du problème

4.1.1 Problème

La problématique de ce chapitre est :

Soit un multiensemble quelconque d'éléments, comment estimer avec une précision raisonnable la cardinalité de cet ensemble, en une seule passe et avec un espace mémoire extrêmement restreint ?

Une première réponse consiste à stocker tous les éléments triés en mémoire de façon unique, et à la fin regarder la longueur de la liste. Cette approche présente deux inconvénients majeurs : tout d'abord, elle demande de stocker tous les éléments distincts, ce qui peut être très coûteux en mémoire, et ensuite elle demande d'insérer tous les éléments de l'ensemble, ce qui est très coûteux en terme de nombre d'accès à une mémoire secondaire, par exemple.

Les algorithmes efficaces qui répondent à cette question se servent de propriétés de distribution de mots aléatoires, et permettent d'évaluer le cardinal de l'oeuvre complète de Shakespeare (5Mb) à quelques pour cents près en utilisant une mémoire de quelques kilobits, cela sans supposer quoi que ce soit sur la structure des éléments. Des résultats analogues peuvent être obtenus pour des ensembles de taille beaucoup plus grande. Par exemple pour des traces d'accès de pages webs, on dispose de très gros ensembles de données qui ont beaucoup de points communs globalement (une ligne est constituée de l'adresse IP du client, de la date, du contenu de la requête) et localement (un même client envoie plusieurs requêtes successivement). L'algorithme Loglog présenté dans ce chapitre ne tient pas compte de ces similitudes parmi les données étudiées, mais seulement des éléments distincts, grâce à l'utilisation de fonctions de hachage.

4.1.2 Motivations

Routeurs. Un routeur internet voit passer un flot continu de paquets, qui sont trop nombreux pour pouvoir être stockés. Estan *et al.* [EVF03] citent des traces d'en-têtes de paquets, produites à un rythme de 0.5GB par heure sous forme compressée ! Ces données ont été collectées pour tracer un ver (Code Red, 1–12 Août 2001), et l'objectif était de compter le nombre de sources infectées distinctes circulant par ce routeur, pour ensuite tenter d'en localiser la source. On voit sur cet exemple l'intérêt crucial de ne pas stocker les données afin d'économiser la mémoire, de faire peu d'opérations à chaque étape, car on doit pouvoir traiter les données à la volée, et enfin on note que l'obtention du résultat exact n'est pas essentielle, qu'on peut se contenter d'une bonne approximation.

De manière plus générale, l'analyse du trafic internet par les routeurs est très demandeuse d'estimations qualitatives du trafic diverses et variées. Estan *et al.* dans [EV01, EVF03] présentent une série d'exemples illustrant cette demande. Pour commencer, certains types d'attaques ("port scan") créent un nombre anormalement élevé d'événements caractéristique au niveau des routeurs, qui peut alors permettre de les détecter. Ensuite cette analyse qualitative peut permettre de bien dimensionner les routeurs, par exemple la taille des différents caches, afin d'optimiser leur utilisation. La figure 4.1, extraite de [EVF03], représente le nombre de connexions distinctes entrantes et sortantes du réseau d'un campus américain. On observe un grand pic de connections qui correspond à une attaque du site.

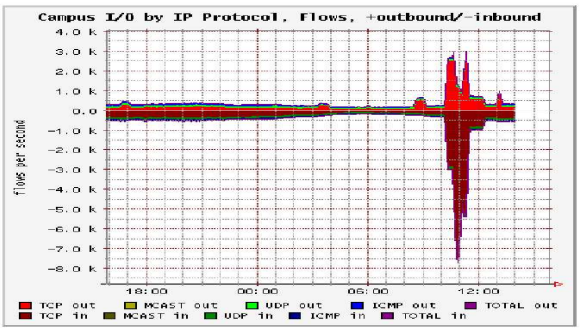

FIG. 4.1 – Evolution du nombre de connections distinctes entrantes et sortantes, le jour d'une tentative d'attaque du site.

Bases de données.

Les estimations probabilistes sont utiles dans la gestion de gros volumes de données. Dans certains cas on peut fournir une réponse approchée à un problème dans un temps beaucoup plus faible que ce qu'il faudrait pour obtenir une réponse exacte. Voir pour cela la description du projet AQUA par Gibbons *et al.* dans [GPA+98].

Un autre exemple, proposé par Palmer *et al.* [PSF+01], décrit l'utilisation d'algorithmes de type comptage probabiliste pour une analyse extensive de la connectivité de la topologie du réseau internet. Un des paramètres étudié est la détermination du nombre de paires de noeuds qui sont à distance au plus h dans le graphe du réseau internet, pour chaque distance h. Étant donné que le graphe de référence possède plus de 300 000 noeuds, le nombre de paires à considérer est largement supérieur à 10^{10}. Cela rend problématique l'utilisation des algorithmes exacts. En pratique, les auteurs de [PSF+01] couplent un algorithme a priori sous-optimal avec des méthodes de comptage probabilistes, et obtiennent de bonnes estimées. Grâce à cela, ils ont pu obtenir des informations extensives sur la métrique du graphe internet en ne gardant en mémoire qu'un très petit sous ensemble des données. Le temps d'exécution a alors été réduit d'un facteur supérieur à 400.

L'estimation de cardinal a de plus des applications en optimisation de requête dans des bases de données. Dans ce domaine, le fait de connaître la cardinalité des ensembles ou sous-ensembles étudiés permet de choisir la stratégie algorithmique la plus adaptée. Pour un problème aussi simple que l'intersection de deux grands ensembles A et B qui contiennent des répétitions, il existe plusieurs manières de procéder. On note a et b le nombre d'éléments contenus dans A et B, et α et β le cardinal (nombre d'éléments distincts) de A et B.

1. On peut trier A, puis chercher chaque élément de B dans cet ensemble, et ne garder que les recherches fructueuses. Cette stratégie a un coût $O(a \log \alpha + b \log \alpha)$. On peut appliquer cette stratégie en inversant les rôles de A et B.
2. On peut trier A, trier B, puis faire l'intersection des deux ensembles triés, ce qui a un coût $O(a \log \alpha + b \log \beta + \alpha + \beta)$.
3. Éliminer les répétitions dans A et/ou B par des méthodes de hachage, puis appliquer l'une ou l'autre des méthodes précédentes. Le coût dépend alors des méthodes de hachage utilisées. Si le hachage se passe dans de bonnes conditions (Tables non pleines) la complexité est $a + b + \alpha \log \alpha + \beta \log \beta$.

On voit alors que le choix de la meilleure stratégie dépend fortement des valeurs de cardinal, mais que les estimations de ces valeurs n'ont en revanche pas besoin d'être extrêmement précises.

4.2 Les réponses apportées - Un survol

Parmi les réponses apportées à ce problème, on distingue dans cette section trois "familles" de méthodes. Une sous-section est ici consacrée à chacune d'elles, afin de présenter les idées et les principaux résultats. L'article de Astrahan, Schkolnick et Whang [ASW87] présente un survol des méthodes disponibles en 1987.

Pour tous ces algorithmes, les données dont on évalue la cardinalité sont des mots binaires de longueur finie. Il est toujours possible de se ramener à ce cas de figure grâce à un *hachage* des données. Selon les cas, et pour faciliter l'exposé, on peut choisir de présenter ces mots binaires de longueur l comme des réels entre 0 et 1 ou comme des entiers entre 0 et $2^l - 1$. (On passe aisément d'une présentation à l'autre par une multiplication par 2^l.)

Dans toute la suite, on suppose que l'ensemble des données distinctes est uniformément réparti parmi les mots binaires ou les réels. Cette hypothèse très réaliste traduit le fait que les données originales sont hachées. Elle est aussi validée a posteriori par les simulations présentées au chapitre 5

On expose tout d'abord l'algorithme de comptage probabiliste, qui est un ancêtre des algorithmes Loglog et super Loglog dont l'étude constitue la majeure partie de ce chapitre, ainsi que du chapitre 5. Le deuxième algorithme discuté est l'échantillonnage adaptatif, dû à Flajolet [Fla90]. Le troisième est basé sur le comptage par collisions, et fait l'objet d'une analyse complète dans la section 4.3, car si cet algorithme est classique, son analyse n'avait pas encore été faite, à ma connaissance. Dans cette section de survol, concernant cet algorithme, seuls les résultats seront énoncés. La dernière sous-section est consacrée à des contributions de Estan, Varghese et Fisk, qui fournissent des algorithmes efficaces, mais qui postulent quelques hypothèses supplémentaires sur l'ordonnancement des données.

4.2.1 Le comptage probabiliste

L'algorithme de comptage probabiliste (*Probabilistic Counting*) présenté par Flajolet et Martin dans [FM85] est un algorithme probabiliste qui estime le nombre d'éléments distincts dans un ensemble en se basant sur les propriétés de fréquence d'apparition de préfixes. Cet algorithme est étudié et étendu par exemple dans [KPS92, KPS96].

La caractéristique à laquelle on s'intéresse est la longueur du préfixe de zéros de chaque mot, ou, autrement dit, la position du premier `1` dans le mot. Si l'ensemble de mots qu'on considère est aléatoire uniformément distribuée, c'est-à-dire que pour chaque mot, chaque bit vaut `0` ou `1` avec une probabilité 1/2, indépendamment des autres, la probabilité que la position du premier 1 soit au moins k dans un mot est 2^{-k}, pour $k \geq 1$ et inférieur à la taille des mots. On note n le nombre de mots distincts dans cet ensemble aléatoire. Le maximum des positions des premiers 1 sur tout l'ensemble a une valeur moyenne proche de $\log_2 n$ (ce résultat est prouvé plus tard dans la section 4.4.2). D'où l'idée d'utiliser ces informations sur les positions des premiers 1 pour estimer $\log_2 n$, et dans un deuxième temps estimer n.

L'algorithme de comptage probabiliste conserve en mémoire un tableau de bits noté T de taille la longueur des mots, et pour chaque mot de l'ensemble de mots étudié, coche la case i si le premier 1 arrive en position i. Finalement $T[i]$ vaut 1 s'il existe un mot dans l'ensemble qui commence par 0^i1. On note R la position du zéro le plus à gauche dans le tableau T. L'article [FM85] montre que l'espérance de R est équivalente à $\log_2 \phi n$, à des fluctuations périodiques négligeables près. La valeur ϕ désigne une constante dont la valeur numérique est $0.77\ldots$. Cependant, l'écart type de cette variable aléatoire est proche de 1.12, ce qui signifie que l'estimation de $\log_2 n$ est relativement imprécise.

L'idée est alors de séparer les mots en différents groupes, d'estimer le nombre d'éléments distincts de chaque groupe, puis de regrouper ces résultats pour réduire la variance. On stocke alors en mémoire m tableaux de bits, $T_0, \ldots, T_{m-1}$. Il est pratique de choisir $m = 2^k$, car alors on peut orienter les mots selon leurs k premiers bits, et aller ensuite chercher la position du premier 1 dans le suffixe. Par exemple, si $k = 2$, le mot 01 001 ... fera cocher la case $T_1[2]$. A chaque tableau de bits T_i on associe la position du zéro le plus à gauche R_i. D'après ce qui précède, on sait que l'espérance de la moyenne empirique des R_i est $\log_2 \phi \frac{n}{m}$, et on pose alors $E_n = \frac{m}{\phi} 2^{\frac{R_0+\cdots+R_{m-1}}{m}}$ comme estimateur de n.

Proposition 4.1 *L'algorithme "comptage probabiliste" fournit une estimation E_n de la cardinalité de l'ensemble qui est asymptotiquement non biaisée, et telle que*

$$\frac{\sigma(E_n)}{n} \simeq \frac{\lambda}{\sqrt{m}} \quad \lambda \approx 0.78,$$

où m est le nombre de mots mémoire utilisés, et oùσ désigne l'écart-type.

On voit que pour une mémoire en bits égale à m fois la longueur des mots, on obtient une erreur standard (définie comme l'écart-type normalisé) de $\frac{0.78}{\sqrt{m}}$. Par exemple pour avoir une erreur standard de 2.5% sur des mots de taille 32 bits, il faut choisir $m = 2^{10}$, ce qui correspond à un espace mémoire de 4ko.

4.2.2 L´échantillonnage adaptatif

L'algorithme d'échantillonnage adaptatif (*Adaptive Sampling*) originellement dû à Wegner est présenté et analysé par Flajolet dans [Fla90]. Cet algorithme stocke en mémoire un échantillon représentatif de mots, et infère ensuite le cardinal de l'ensemble à partir du cardinal de l'échantillon et des conditions d'échantillonnages.

On dispose d'un espace mémoire de m mots, c'est-à-dire de taille m fois la longueur d'un mot. A chaque instant, on stocke tous les mots qui satisfont le critère de filtrage. Au départ, ce critère est le prédicat universellement vrai, c'est-à-dire que tous les mots le satisfont. Ensuite lorsque la mémoire devient pleine, on durcit le critère de filtrage, afin d'éliminer une partie des éléments mémorisés, et on continue. Un critère de filtrage simple consiste à sélectionner tous les mots commençant par k zéros, la valeur de k étant 0 au départ, et augmentant de 1 à chaque fois que la mémoire sature. Au moment d'estimer le cardinal de l'ensemble total, on considère que les mots commençant par le préfixe 0^k sont représentatifs, et si ν désigne le nombre de mots en mémoire, l'algorithme répond $\nu 2^k$.

Proposition 4.2 *L'algorithme d'échantillonnage adaptatif fournit une estimation E_n de la cardinalité de l'ensemble qui est non biaisée et telle que*

$$\frac{\sigma(E_n)}{n} \simeq \frac{\lambda}{\sqrt{m}} \quad \lambda \approx 1.20,$$

où σ désigne l'écart-type.

Cet algorithme est moins performant a priori que le comptage probabiliste, mais présente l'avantage d'être non biaisé non seulement en un sens asymptotique mais aussi pour des ensembles de petite taille.

4.2.3 Les méthodes de comptage par collisions

Le hachage est une transformation qui permet d'indexer les éléments d'un ensemble. Soit un ensemble de données $\mathcal{A}$, une fonction de hachage est une fonction de $\mathcal{A}$ vers les entiers inférieurs à m. Ceci permet d'associer à un élément de l'ensemble une case de la table de hachage, qui est un tableau de taille m. Ensuite selon les applications, différentes options peuvent être choisies. Pour l'indexation les éléments seront stockés (par exemple dans une liste chaînée) dans la case qui leur est associée. Ceci permet des recherches efficaces, à condition que les éléments se répartissent de façon équitable dans la table. D'autres stratégies ne permettent que le stockage d'un nombre borné d'éléments dans chaque case et il faut alors trouver une méthode pour résoudre les débordements. L'algorithme de hachage à essai linéaire essaye d'insérer l'élément dans la case suivante, voir le travail de Flajolet, Poblete et Viola [FPV98, VP98]. Le hachage à essai aléatoire consiste à tirer une case au hasard jusqu'à ce qu'il y ait une place libre. Le hachage à essai uniforme associe à chaque élément une permutation des cases, et cherche une place vide dans la table de hachage dans l'ordre de la permutation, cet algorithme est présenté par Larson dans [Lar83]. Ces stratégies ont été présentées en détail dans le chapitre 3 dont le sujet est l'étude de l'algorithme de hachage à essais aléatoires. Pour une étude générale voir [Knu98]. Le hachage permet aussi des tris très efficaces, en temps moyen linéaire, si l'on connaît a priori des informations sur la distribution des éléments [Knu98].

Le comptage par collisions. Ce paragraphe présente l'algorithme d'estimation de cardinal basé sur le comptage par collisions, ainsi que les principaux résultats. Pour l'analyse complète, voir la section 4.3. Cet algorithme, présenté par Whang, Vander-Zanden et Taylor dans [WVZT90] est basé sur une fonction de hachage notée h. On dispose d'une table de bits de taille m, nommée T. La fonction de hachage h associe par hypothèse à chaque élément un entier entre 0 et $m-1$ avec probabilité uniforme.

```
Algorithme de comptage par collisions.

Pour tout i : T[i]:=0;
Pour tout élement x, T[h(x)]:=1;
Soit V=#{T[i]=0}   \\nombre de cases égales à 0

 Si V différent de 0 Alors Retourner -mlog(V/m)
                     Sinon Retourner -mlog(1/m)+m
```

On note $E_n := -m\log(V/m)$ l'*estimateur* si le cardinal, inconnu, de l'ensemble étudié est n. On a alors les résultats suivants sur l'espérance et la variance de E_n, notées respectivement $\mathbb{E}$ et $\mathbb{V}$.

Proposition 4.3 *On note μ et σ^2 l'espérance et la variance du nombre de cases vides V. Ces quantités s'expriment de la façon suivante, avec $\rho = n/m$ fixé*

$$\mu = m\left(1-\frac{1}{m}\right)^n \sim me^{-\rho} \quad et$$

$$\sigma^2 = m(m-1)\left(1-\frac{2}{m}\right)^n + m\left(1-\frac{1}{m}\right)^n - m^2\left(1-\frac{1}{m}\right)^{2n} \sim m\left(e^{-\rho}-e^{-2\rho}\right).$$

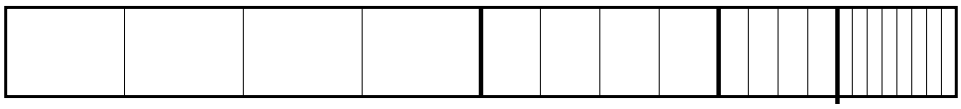

FIG. 4.2 – Un exemple de table de hachage multirésolution.

Lorsque n est inférieur à cm, où c est une constante, l'espérance et la variance de l'estimateur E_n admettent les développements

$$\mathbb{E}(E_n) = n + m\frac{\sigma^2}{2\mu^2} + o\left(m\frac{\sigma^2}{\mu^2}\right),$$

$$\mathbb{V}(E_n) \sim m^2\frac{\sigma^2}{\mu^2}.$$

Tant que n est proportionnel à m, ces développements se récrivent $\mathbb{E}(E_n) \sim n + \frac{e^{n/m}-1}{2}$, et $\mathbb{V}(E_n) \sim e^{n/m} - 1$. On constate sans surprise que le biais de l'algorithme augmente lorsque n devient grand et que la variance augmente également avec n.

4.2.4 Les tables adaptatives

Cette section s'intéresse à des algorithmes proposés par Estan,Varghese et Fisk [EVF02]. Ces algorithmes sont inspirés de l'algorithme de comptage par collisions, mais contrairement à celui ci, fournissent des résultats pour des valeurs de n grandes. L'idée commune à cette gamme d'algorithmes est de faire évoluer la granularité de la table de hachage. Cette stratégie repose sur l'hypothèse que les éléments sont répartis de façon "relativement uniforme". Cette restriction empêche cette famille d'algorithmes d'être universelle, cependant, sur certaines données réelles comme des traces de routeurs, les résultats obtenus sont compétitifs.

Table adaptative. L'algorithme de la table adaptative (adaptive bitmap) est une méthode qui est basée sur le hachage et les estimations de collisions, avec des paramètres évoluant dynamiquement de sorte à toujours utiliser des tables de hachage raisonnablement pleines, ce grâce à un bon choix d'échelle. Cette méthode est développée par Cristian Estan, Georges Varghese et Mike Fisk dans [EVF02, EVF03]. D'autres problématiques de comptage sont étudiées avec des méthodes similaires dans [EV02].

Les différents algorithmes basés sur ce type de méthodes qui sont présentés dans [EVF03] sont utilisés en pratique pour résoudre un certain nombre de problèmes, parmi lesquels l'analyse qualitative de flots internet au niveau des routeurs, la détection systématique de scan de ports, ou encore l'estimation de taux de diffusion de virus le long du réseau. Pour toutes ces applications, ils ont permis des gains en mémoire extrêmement significatifs.

Table multirésolution. L'algorithme d'estimation de cardinal le plus simple est le comptage par collisions. Cet algorithme présente le sérieux inconvénient de devenir totalement inadapté lorsque le cardinal à estimer devient trop grand. Pour pallier ce problème Estan *et al.* proposent d'utiliser une table multirésolution (multirésolution bitmap) qui est une table de hachage dont la résolution est variable selon la zone dans laquelle on se trouve. Formellement, la résolution d'une zone est définie comme la densité de ses cases.

La figure 4.2 propose un exemple avec quatre niveaux de résolution. Sur cette figure, les précisions relatives des quatre zones sont dans les rapports 1, 2, 4 et 8. Ceci permet, après coup de choisir la zone qui avait la bonne résolution, et d'estimer le cardinal recherché grâce au taux de remplissage de cette partie (en ajoutant éventuellement une influence des zones voisines). En pratique, cette méthode impose une limitation assez restrictive sur la taille du cardinal à estimer, car au delà de cette limite, aucune zone ne possède la résolution adéquate, et la table est entièrement pleine.

Table multirésolution adaptative. La méthode utilisée en pratique est un mélange entre les tables adaptatives et les tables multirésolution, laquelle est présentée maintenant.

Pour résoudre le problème de "zone de validité" trop petite et fixée par avance, Estan *et al.* ont proposé un algorithme basé sur des tables multirésolution adaptatives. Remarquons tout d'abord que l'algorithme basé sur une table multirésolution tel qu'on l'a présenté dans le paragraphe précédent maintient une seule table de hachage, mais d'un point de vue théorique on peut considérer qu'il se base sur plusieurs tables de hachage, chacune avec une résolution différente, et toutes disjointes. On suppose dans la suite que les éléments à étudier sont lus comme des entiers entre 0 et 1. De manière générale, on dispose de m tables de hachage, $H_1, \ldots, H_m$, toutes couvrant un intervalle distinct (par exemple, au départ, la table H_i couvre l'intervalle $[2^{-i}, 2^{-i-1}]$), et de résolution de plus en plus précise (par exemple on peut doubler la résolution entre deux tables consécutives).

L'idée de la table multirésolution adaptative est que lorsque la première table est d'une résolution trop grossière, et ne peut plus être utilisée pour donner d'information, alors on cesse de la maintenir, on décale toutes les tables (c'est à dire $H_i := H_{i+1}$), et on crée une nouvelle table H_m qui est d'une résolution plus précise que H_{m-1} et couvre, par exemple, l'intervalle $[2^{-m}, 2^{-m+1}]$. Ainsi l'intervalle de validité se décale au fur et à mesure des besoins, et cette adaptabilité permet d'estimer le cardinal de l'ensemble étudié avec une zone de validité beaucoup plus grande.

Cet algorithme présente le défaut de nécessiter pour sa validité des hypothèses particulières portant sur la distribution des éléments. Même si ces hypothèses sont vérifiées dans un certain nombre d'applications, elles empêchent cet algorithme d'être une solution universelle au problème d'estimation de cardinalité. Le problème vient du fait que lorsqu'on décale d'un cran les tables de hachage, la table H_m se met à recueillir des informations sur un intervalle qui jusque là avait été négligé. Les résultats de cet algorithme ne sont donc valables que si ce qui se passe dans la zone nouvellement couverte par H_m est "représentatif" de ce qui s'y passait avant, c'est-à-dire si le flot de données est en un certain sens "homogène".

Résultat 4.1 *A partir de résultats expérimentaux et d'une analyse heuristique, on obtient les résultats suivants.*

L'algorithme d'estimation de cardinal basé sur une table de hachage multirésolution adaptative renvoie une estimation non biaisée lorsque les données sont distribuées de façon homogène. De plus la précision attendue est

$$\textit{Précision} \approx \frac{0.9}{\sqrt{m}},$$

où la taille de la mémoire auxiliaire utilisée est $m \log n_{max}$*, avec* n_{max} *la limite de validité de l'algorithme.*

En conclusion ce type de méthode fournit d'excellents résultats pratiques pour des applications qui présentent une "bonne" répartition des données. Par exemple, une table multirésolution adaptative permet d'évaluer le nombre de flots distincts sur un lien internet en utilisant une mémoire de 2Ko avec une erreur moyenne de moins de 1%, tant que le nombre de flots distincts est inférieur à 100 millions [EVF02].

4.3 Comptage par collisions - Analyse

Cette section présente l'analyse de l'algorithme de comptage par collisions déjà introduit dans la section 4.2.3 et qu'on rappelle brièvement. On dispose d'une fonction de hachage h uniforme et d'une table de bits de taille m, dans laquelle on hache un ensemble. Au début de l'algorithme, toutes les cases de la table sont initialisées à 0. Ensuite, pour chaque élément x de l'ensemble, la table est mise à jour en mettant la valeur de la case $h(x)$ égale à 1. A la fin, la case i vaut 1 s'il existe x tel que $h(x) = i$, et 0 sinon. On observe que l'état de la table de bits à la fin de l'algorithme ne dépend que de l'ensemble des éléments distincts, et pas des répétitions éventuelles.

Pour obtenir une estimation de n le cardinal (inconnu) de cet ensemble, on se base sur le remplissage de la table de bits. La variable aléatoire qui code le nombre de cases vides dans la table après hachage d'un ensemble de cardinal n est notée V_n.

4.3.1 Choix de l'estimée

Pour étudier la distribution de V_n, on fait appel au formalisme des urnes, lequel permet d'obtenir directement les résultats recherchés. On considère qu'un élément haché est une boule lancée, et que les cases de la table de hachage sont des urnes. La probabilité qu'il y ait k cases vides est notée $p_{n,k} := \Pr(V_n = k)$.

Lorsque n boules sont lancées dans m urnes avec probabilité uniforme, la probabilité $p_{n,k}$ qu'il y ait k urnes vides est

$$p_{n,k} = \frac{m!}{k!m^n} \left\{ {n \atop m-k} \right\} = \frac{n!}{m^n} [u^k][z^n](e^z - 1 + u)^m,$$

où la notation $\left\{ {n \atop k} \right\}$ désigne les nombres de Stirling de deuxième espèce, voir [AS73, GKP94] pour la définition de ces nombres et leurs propriétés. La série génératrice de la variable aléatoire est

$$\sum p_{n,k} u^k z^n / n! = (e^{z/m} - 1 + u)^m.$$

La variable z code le nombre d'éléments jetés dans la table, et u le nombre de cases vides. Pour comprendre cette série génératrice, on observe simplement que chaque urne est soit vide soit non vide. Si une urne est vide alors elle est codée par $z^0 u$ qui marque le fait qu'il y ait une urne vide (u) qui contient zéro éléments (z^0). Et si au contraire une urne est non vide, alors on la marque par $u^0(e^{z/m} - 1)$, car cette urne ne compte pas comme urne vide, et contient au moins un élément. Le coefficient $1/m$ dans l'exponentielle est la normalisation qui permet d'obtenir une série génératrice de probabilité. La série génératrice s'obtient alors en élevant $e^{z/m} - 1 + u$ à la puissance m pour prendre en compte le fait que les boules se distribuent parmi m urnes.

La moyenne du nombre de cases vides s'obtient facilement en dérivant la série génératrice $(e^{z/m} - 1 + u)^m$ par rapport à u (ce qui vaut $\sum k p_{n,k} u^k z^n / n!$) puis en l'évaluant en $u = 1$ (on obtient $\sum k p_{n,k} z^n / n!$), et finalement en extrayant le coefficient de z^n et en multipliant le tout par $n!$. On obtient alors que l'espérance de V_n notée $\mu(n)$ vaut

$$\mu(n) = m \left(1 - \frac{1}{m}\right)^n.$$

Cette valeur est approchée par $me^{-\rho}$, avec ρ qui est le taux de remplissage de la table c'est-à-dire $\rho = n/m$.

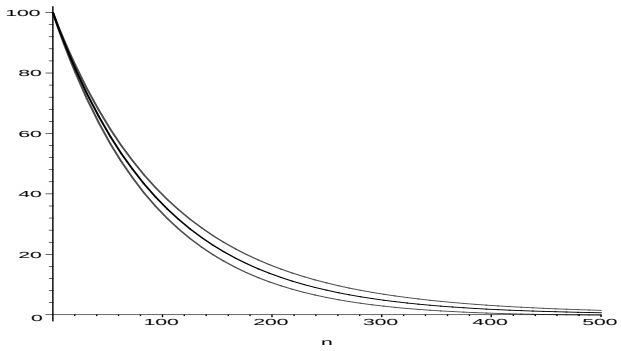

FIG. 4.3 – La moyenne plus ou moins l'écart type du nombre de cases vides dans la table de hachage.

La variance, notée $\sigma^2(n)$ est obtenue par la formule usuelle $f''(1)+f'(1)-(f'(1))^2$ où f désigne la série génératrice des $p_{n,k}$, soit

$$\sigma^2(n) = m(m-1)\left(1-\frac{2}{m}\right)^n + m\left(1-\frac{1}{m}\right)^n - m^2\left(1-\frac{1}{m}\right)^{2n}.$$

La variance peut être approchée par $m\left(e^{-\rho}-e^{-2\rho}\right)$ lorsque n et m tendent simultanément vers l'infini. Lorsque la valeur de n n'est pas explicitement nécessaire, on note $\mu(n)$ et $\sigma(n)$ respectivement μ et σ. La figure 4.3 montre la courbe $y=\mu(n)$, pour $m=100$, entourée des courbes $y=\mu(n)\pm\sigma(n)$.

L'estimation de la valeur moyenne de V_n, soit $\mu(n) \sim me^{-n/m}$, suggère de choisir comme estimateur de n :

$$E_n := -m\log\frac{V_n}{m}.$$

Ceci fournit l'algorithme d'estimation de cardinal de comptage par collisions qui a été annoncé dans la section 4.2.3. Le cas particulier où la table de hachage est pleine doit être traité séparément car le logarithme n'est alors plus défini. La solution choisie ici est d'utiliser l'estimation pour le cas où la table est presque pleine ($V_n = 1$) et de considérer qu'il faut environ m essais pour la remplir entièrement. L'estimation renvoyée pour le cas $V_n = 0$ est alors $m\log m + m$

Le reste de la section est consacré à l'étude de la variable aléatoire E_n, en commençant par la distribution de V_n.

4.3.2 Description du nombre de cases vides

L'étude de V_n se fait en deux temps. Tout d'abord on montre que V_n peut être approximée par une gaussienne au voisinage de sa moyenne μ, et on trouve ensuite une majoration à décroissance exponentielle en dehors de cette zone. Comme l'essentiel de la distribution est située autour de la moyenne, ainsi qu'on le voit sur la figure 4.4, cette approximation gaussienne peut être utilisée pour l'étude de E_n.

V_n est gaussienne. On pose $f(z,u)=(e^z-1+u)^m$ qui est à normalisation près la série génératrice de probabilité de V_n. L'objectif est d'estimer le coefficient de z^n dans $f(z,u)$. On se propose de procéder par méthode de col appliquée aux grandes puissances. Cette méthode est détaillée par Drmota et Soria dans [DS95]. On a

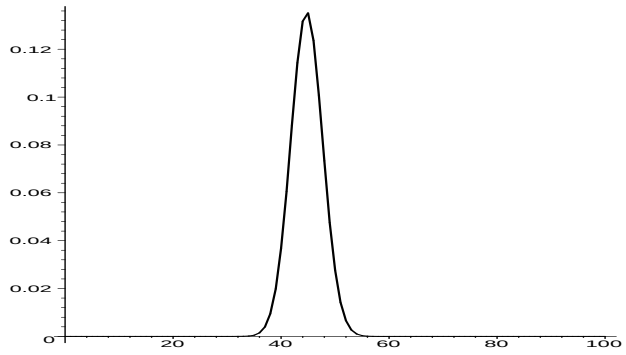

FIG. 4.4 – Distribution de V_{80}, avec $m = 100$.

$$[z^n]f(z,u) = \frac{1}{2i\pi}\oint \frac{(e^z - 1 + u)^m}{z^n}\frac{dz}{z}.$$

Ici l'intégrale est prise le long d'un contour entourant 0. On pose $h(z,u) = \log(e^z - 1 + u) - \frac{n}{m}\log z$, ce qui donne

$$[z^n]f(z,u) = \frac{1}{2i\pi}\oint e^{mh(z,u)}\frac{dz}{z}.$$

Par la méthode classique de col, on obtient que

$$[z^n]f(z,u) \sim \frac{e^{\zeta(u)-1+u}}{\zeta(u)^n}\frac{1}{\zeta(u)\sqrt{2\pi h''(\zeta(u))m}} = \lambda(u)^n B(u),$$

où $\zeta(u)$ est le point col, défini par l'équation implicite $\zeta(u)e^{\zeta}(u)/(e^{\zeta}(u) - 1 + u) = \rho$. Cette approximation est uniforme pour u dans un voisinage compact de 1. Pour une preuve détaillée voir l'article de Drmota [Drm94b].

Par le théorème des quasi puissances de Hwang [Hwa94], ou en anglais dans [FS05], on conclut ensuite au fait que la loi de V_n est asymptotiquement gaussienne.

On obtient par une seconde application de la méthode de col l'équivalence $p_{n,k} \sim e^{-\frac{(k-\mu)^2}{2\sigma^2}}\frac{1}{\sigma\sqrt{2\pi}}$, lorsque k est situé dans un voisinage de μ de rayon $b_{n,m} := \sigma\log^{\alpha} m$, $\alpha < 1/2$. Ceci va permettre d'étudier E_n dans la région centrale de cette gaussienne.

Grandes déviations. Pour le cas où k n'est pas situé près de μ, on peut utiliser la méthode de col pour obtenir des bornes de grandes déviations de $p_{n,k}$. On obtient qu'il existe C_1, C_2 tels que

$$p_{n,k} \leq C_1(C_2)^{|\mu-k|} \quad \text{avec } C_2 < 1.$$

Preuve.

On dispose de la majoration $p_{n,k} \leq \frac{n!}{m^n}\frac{\lambda(u_0)^n B(u_0)}{u_0^k}$. Lorque u_0 est inférieur à 1, cela montre une décroissance exponentielle des coefficients $p_{n,k}$, pour $k < \mu$.

Le cas où k est plus grand que la moyenne est analogue, et traité de la même manière. ■

4.3.3 Analyse de l'espérance de l'estimateur E_n

Le calcul de l'espérance de E_n permet de déterminer si l'estimateur est bon, et éventuellement modifier le choix de l'estimateur initial, afin d'en obtenir un nouveau qui soit non biaisé.

Proposition 4.4 *L'espérance de E_n admet le développement asymptotique*

$$\mathbb{E}E_n = n + m\frac{\sigma^2}{2\mu^2} + o\left(m\frac{\sigma^2}{2\mu^2}\right),$$

lorsque n est inférieur à cm, où c est une constante arbitraire fixée.

Preuve. L'espérance de E_n s'écrit $\sum_{k\geq 1} -m\log\frac{k}{m}p_{n,k} + m(\log m + 1)p_0$. Le cas particulier $k = 0$ n'est en fait pas essentiel, car, tant que n est proportionnel à m, la probabilité p_0 est exponentiellement négligeable. Ceci provient par exemple de l'équivalence $p_0 \sim 1 - (1 - e^{-\rho})^m$. Dans la suite, la notation $\sum_a^b$, où a et b ne sont pas nécessairement entiers, représente $\sum_{\lceil a\rceil}^{\lfloor b\rfloor}$, c'est-à-dire une somme sur tous les entiers de l'intervalle $[a, b]$.

On applique un changement de variable $k := k - \mu$ pour centrer la variable aléatoire, et on développe le logarithme par $\log\frac{k+\mu}{m} = \log\frac{\mu}{m} + \log(1 + \frac{k}{\mu})$ ce qui donne

$$\mathbb{E}E_n = n - \sum_{k=-\mu+1}^{m-\mu} m\log(1+\frac{k}{\mu})p_{n,k+\mu}.$$

Dans cette équation, le premier terme n est le terme principal, et le terme somme apporte des corrections. Pour chacun des termes de cette somme, c'est la probabilité $p_{n,k}$ qui dicte le comportement général tandis que le logarithme n'apporte qu'une légère modification. En particulier le comportement des probabilités rend les termes de la somme qui ne sont pas proches de $k = 0$ négligeables. On découpe donc la somme en trois parties, la partie centrale autour de $k = 0$ et les deux extrémités. On écrit alors

$$\sum_{k=-\mu+1}^{m-\mu} = \sum_{k=-\mu+1}^{-b_{n,m}} + \sum_{k=-b_{n,m}}^{b_n,m} + \sum_{k=b_{n,m}}^{m-\mu} = S_1 + S_2 + S_3,$$

où $b_{n,m} = \sigma\log^\alpha m$ est la limite de la zone où l'approximation gaussienne a lieu.

On s'intéresse tout d'abord à la somme S_2 qui est la plus importante, et sur laquelle on peut appliquer l'équivalence $p_{n,k} \sim e^{-\frac{(k-\mu)^2}{2\sigma^2}}\frac{1}{\sigma\sqrt{2\pi}}$. Sur cette zone de sommation, on peut développer le logarithme en série entière, et ensuite intervertir les deux sommes (par exemple grâce au théorème de Fubini) pour aboutir au développement suivant :

$$S_2 \sim \sum_{i\geq 1} m\frac{(-1)^{i-1}}{i\sqrt{2\pi\sigma^2}} \sum_{k=-b_{n,m}}^{b_{n,m}} \frac{k^i}{\mu^i}e^{-\frac{k^2}{2\sigma^2}}.$$

On remarque tout d'abord que lorsque l'entier i est impair, pour des raisons de symétrie, le terme $\sum_{k=-b_{n,m}}^{b_{n,m}} \frac{k^i}{\mu^i}e^{-\frac{k^2}{2\sigma^2}}$ est nul, la somme S_2 est donc restreinte aux i pairs. Pour étudier cette somme, on énonce le lemme suivant.

Lemme 4.1 *Soit $K_i := \sum_{k=-b_{n,m}}^{b_{n,m}} \frac{k^{2i}}{\mu^{2i}}e^{-\frac{k^2}{2\sigma^2}}\frac{1}{\sigma\sqrt{2\pi}}$, alors K_i est une suite décroissante, de décroissance géométrique $K_{i+1} \leq \frac{K_i}{4}$ et on a de plus $K_1 \sim \frac{\sigma^2}{\mu^2}$. $K_2 \sim 3\frac{\sigma^4}{\mu^4}$ et $K_3 = O(\frac{\sigma^6}{\mu^6})$.*

Preuve du lemme. La décroissance géométrique se prouve grâce à la majoration $(k/\mu)^2 \leq 1/4$, et les équivalents pour K_1, K_2 et K_3 se prouvent en montrant que $\sum_{k=-b_{n,m}}^{b_{n,m}} \frac{k^i}{\mu^i} e^{-\frac{k^2}{2\sigma^2}} \sim \int_{-\infty}^{\infty} \frac{x^i}{\mu^i} e^{-\frac{x^2}{2\sigma^2}} dx$, pour $i = 1, 2, 3$, et en calculant l'intégrale.

Ce lemme permet d'écrire que $S_2 = \sum_i m \frac{-K_i}{2i}$. La preuve de la proposition 4.4 dépend alors uniquement des majorations des sommes S_1 et S_3.

La somme S_1 est majorée facilement grâce aux observations suivantes. Lorsque $-\mu + 1 \leq k \leq -b_{n,m}$ alors on a les majorations suivantes $|\log(1+k/\mu)| \leq \log \mu$, et $p_{n,k} \leq C_1 (C_2)^{b_{n,m}}$, ce qui donne finalement pour S_1

$$S_1 \leq m^2 \log \mu C_1 (C_2)^{b_{n,m}}.$$

Ce qui prouve la décroissance exponentielle de S_1 lorsque n est proportionnel à m. On montre de façon exactement identique la décroissance de la somme S_3.

Il est possible d'obtenir un développement de $\mathbb{E}E_n$ avec plus de termes, comme le suggère le lemme 4.1, à condition d'avoir une estimation plus précise des probabilités $p_{n,k}$.

■

Remarque. Lorsque la table de hachage est pleine, l'estimation du nombre d'éléments renvoyée par l'algorithme est équivalente à $m \log m$. Ceci n'est pas une surprise, car cette valeur apparaît déjà dans le problème du collectionneur de coupons qui est présenté par exemple par Motwani et Raghavan dans [MR95], et rappelé dans la section 3.4.1. Le collectionneur de coupons achète des tablettes de chocolat Poulain, et collectionne les vignettes. On suppose que les vignettes sont toutes équiprobables, combien de tablettes devra manger en moyenne le collectionneur avant d'avoir un album plein ? La réponse est équivalente à $m \log m$ où m est le nombre de vignettes. Le problème du collectionneur de coupons est le même que le comptage par collisions dans une table de hachage : il suffit de considérer que chaque vignette Poulain est représentée par une urne, et que chaque tablette de chocolat est un élément haché dans la table. La table de hachage est pleine lorsque la collection est complète (ou réciproquement), et ce au bout d'un temps moyen équivalent à $m \log m$.

4.3.4 Analyse de la variance de E_n

Le moment d'ordre 2 de E_n est égal à $M_2 := \sum_k (m \log(k/m))^2 p_{n,k}$, ce qui permet d'écrire la variance de E_n sous la forme

$$\mathbb{V}E_n = \sum_k (m \log(k/m))^2 p_{n,k} - \left(\sum_k -m \log(k/m) 2 p_{n,k} \right)^2.$$

On peut énoncer la propriété suivante

Proposition 4.5 *Lorsque n est plus petit que cm, où c est une constante fixée, on a*

$$\mathbb{V}E_n \sim m^2 \frac{\sigma^2}{\mu^2}.$$

Preuve. On s'intéresse ici à trouver un équivalent de M_2. Pour cela, on utilise les approximations et majorations de $p_{n,k}$ montrées dans la sous-section précédente, et comme pour l'espérance, on coupe la somme en trois parties, après l'avoir recentrée.

$$M_2 = \sum_{k=-\mu+1}^{m-\mu} (m \log((k+\mu)/m))^2 p_{k+\mu} = \sum_{k=1-\mu}^{-b_{n,m}} + \sum_{-b_{n,m}}^{b_{n,m}} + \sum_{b_{n,m}}^{m-\mu} = S_1 + S_2 + S_3.$$

Pour la partie centrale S_2, on applique l'approximation gaussienne de $p_{n,k}$.

$$S_2 \sim \sum_{-b_{n,m}}^{b_{n,m}} (m \log((k+\mu)/m))^2 e^{-\frac{k^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2}}$$

Comme pour l'espérance, on développe le logarithme, $\log((k+\mu)/m) = \log(\mu/m) + \log(1+k/\mu)$. En développant ensuite le carré autour du logarithme et en utilisant les résultats de la section précédente on obtient

$$S_2 = n^2 + 2nm\left(\frac{\sigma^2}{2\mu^2} + o\left(\frac{\sigma^2}{\mu^2}\right)\right) + m^2 \sum_{-b_{n,m}}^{b_{n,m}} \log^2(1+k/\mu) e^{-\frac{k^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2}}.$$

On appelle A la somme dans l'équation ci dessus. Dans cette somme, on développe le logarithme, en ne gardant que les premiers termes de puissance paire (par symétrie les termes de puissances impaires n'apportent aucune contribution). On obtient alors $A \sim \frac{\sigma^2}{\mu^2}$. Lorsqu'on compile tous ces résultats, on obtient un équivalent de S_2

$$S_2 = n^2 + 2nm\left(\frac{\sigma^2}{2\mu^2} + o\left(\frac{\sigma^2}{\mu^2}\right)\right) + m^2\left(\frac{\sigma^2}{\mu^2} + o\left(\frac{\sigma^2}{\mu^2}\right)\right).$$

La somme S_1 est majorée grâce aux résultats de grandes déviations obtenus sur $p_{n,k}$, et à des majorations grossières sur le reste. Ces résultats permettent d'écrire

$$S_1 \leq m^3 \log^2 m C_1 (C_2)^{\sigma \log^\alpha m},$$

qui montre une décroissance exponentielle de S_1. La somme S_3 est majorée de manière parfaitement similaire, et on a ainsi prouvé que M_2 est équivalent à S_2. Il n'y a alors plus qu'à regrouper tous les développements obtenus pour conclure que

$$\mathbb{V}E_n \sim m^2\left(\frac{\sigma^2}{\mu^2}\right),$$

ce qui achève la preuve de la proposition. ■

4.3.5 Algorithme

Les résultats de la proposition 4.4 montrent que l'algorithme de comptage par collisions classique est biaisé. Le fait de connaître le biais permet de développer un nouvel algorithme, basé sur les mêmes idées, mais qui corrige ce biais. Pour obtenir un algorithme non biaisé, il suffit a priori d'inverser la formule de la proposition 4.4. Cela n'est pas directement réalisable, mais une manière d'approcher cette inversion est de procéder par raffinement successifs.

On rappelle la relation entre l'espérance de E_n et n :

$$\mathbb{E}(E_n) = n + m\frac{\sigma^2(n)}{2\mu^2(n)} + o\left(m\frac{\sigma^2(n)}{\mu^2(n)}\right),$$

où

$$\mu(n) \sim me^{-\rho} \quad \sigma^2(n) \sim m\left(e^{-\rho} - e^{-2\rho}\right) \quad \rho = \frac{n}{m}.$$

L'algorithme consiste à obtenir une valeur approchée de n par la méthode classique $n_0 = E_n$, puis raffiner cette valeur en approchant $\mu(n)$ et $\sigma(n)$ par $\mu(n_0)$ et $\sigma(n_0)$.

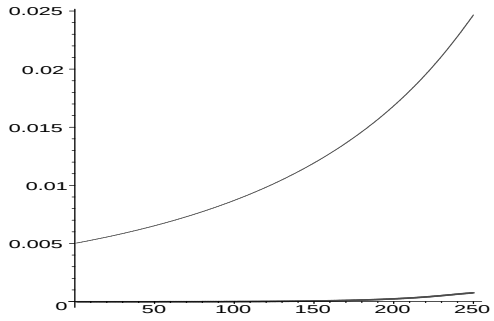

FIG. 4.5 – Comptage par collisions classique versus comptage par collisions raffiné, pour $m = 100$, et $n \leq 250$. Les courbes représentent les valeurs de $|E_n/n - 1|$, où E_n est l'estimateur, soit de l'algorithme classique [trait fin], soit de l'algorithme raffiné [trait épais].

```
Algorithme de comptage par collisions (II)

Pour tout i : T[i]:=0;
Pour tout élement x, T[h(x)]:=1;
Soit V=#{T[i]=0}   \\nombre de cases égales à 0
Si V différent de 0 Alors n0=-mlog(V/m)
                    Sinon n0=-mlog(1/m)+m
mu:=mexp(-n0/m); sigma2:= m(exp(-n0/m)-exp(-2n0/m));
Renvoyer n1:=n0-msigma2/(2mu^2);
```

Cet algorithme renvoie une première valeur raffinée de n. La précision pourrait être améliorée en se basant sur un développement de l'espérance plus long, et plus d'étapes de raffinement. La figure 4.5 montre le comportement de l'algorithme de comptage par collisions (II) par rapport au comptage par collisions classiques. On observe que le niveau de raffinement choisi dans le programme ci-dessus est déjà très bon.

4.4 L'algorithme "Loglog counting"

L'algorithme "Loglog counting" [DF03] permet d'estimer efficacement, rapidement et avec très peu de mémoire auxiliaire le nombre d'éléments distincts dans un grand ensemble de données. C'est un algorithme probabiliste, dont la précision attendue est fonction de la mémoire utilisée. Par exemple, avec l'espace mémoire utilisé par la Figure 4.4 (256 caractères utilisant chacun 5 bits), on évalue le cardinal de l'oeuvre complète de Shakespeare avec une précision de 9.4%.

```
ghfffghfghgghgggggghghheehfhfhhgghghghhfgffffhhhiigfhhffgfiihfhhh
igigighfgihfffghigihghigfhhgeegeghggghhhgghhfhidiigihighihehhhfgg
hfgighigffghdieghhhgggghhfghhfiiheffghghihifgggffihgihfggighgiiif
fjgfgjhhjiifhjgehgghfhhfhjhiggghghihigghhihihgiighgfhlgjfgjjjmfl
```

FIG. 4.6 – Espace mémoire utilisé pour estimer le nombre de mots différents dans l'oeuvre complète de Shakespeare (disponible à `http ://the-tech.mit.edu/Shakespeare/`). L'estimée obtenue ici est $n^\circ = 30897$, et le nombre exact est $n = 28239$.

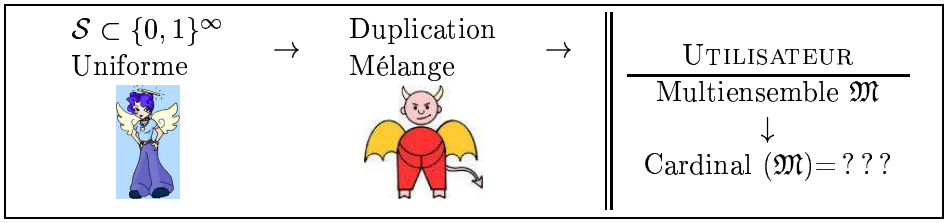

FIG. 4.7 – Un sous-ensemble de $\{0,1\}^\infty$ est choisi uniformément (par l'ange) puis dupliqué et mélangé arbitrairement (par le démon) et enfin remis à l'utilisateur qui doit estimer son cardinal

4.4.1 Énoncé du problème

Le problème posé est d'évaluer le cardinal d'un multiensemble $\mathcal{U}$. Pour cela, on suppose encore disposer d'une fonction de hachage h, qui transforme les éléments de $\mathcal{U}$ en des chaînes de bits de taille suffisamment grande et telles que les bits de la valeur hachée aient l'apparence d'un aléa uniforme et indépendant. Cette supposition est justifiée par Knuth qui écrit dans [Knu98] : "*It is theoretically impossible to define a hash function that creates random data from non-random data in actual files. But in practice it is not difficult to produce a pretty good imitation of random data.*"

En se basant sur cette supposition, le problème est alors modélisé de la façon suivante.

- Soit $\mathcal{B} = \{0,1\}^\infty$ l'ensemble des mots possibles. Un *multiensemble idéal* $\mathfrak{M}$ de cardinal n est construit comme suit. On pioche tout d'abord n mots de façon aléatoire dans $\mathcal{B}$, avec probabilité uniforme (i.e. chaque lettre de chaque mot est indépendante de toutes les autres et vaut 0 ou 1 avec probabilité 1/2). Puis on réplique certains éléments de façon arbitraire et un nombre de fois quelconque. Enfin on applique une permutation arbitraire.
- On dispose alors du multiensemble idéal $\mathfrak{M}$ (potentiellement très grand), et le but est de déterminer la valeur (inconnue) de n, avec un coût mémoire très faible et peu de calculs. On souligne le fait qu'aucune hypothèse n'est faite sur la nature et le nombre des réplications, ainsi que sur la permutation finale (qui peut par exemple trier les éléments).
- Ce processus est illustré par la Figure 4.7.

Le fait de considérer des mots infinis facilite la présentation pour l'instant. L'algorithme portant sur des mots finis ainsi que ses performances seront discutés plus tard, à la section 5.4.

4.4.2 Intuition

Dans cette sous-section, on s'intéresse à des mots binaires infinis sur l'alphabet $\{0,1\}$. Pour un mot aléatoire, c'est-à-dire dont chaque bit est 0 ou 1 avec probabilité 1/2, la probabilité de commencer par au moins k zéros vaut 2^{-k}. Par conséquent, la probabilité de commencer par moins de k zéros est égale à $1-2^{-k}$. Lorsqu'on considère un ensemble de n mots binaires infinis aléatoires indépendants, la probabilité que le plus grand préfixe de zéros soit de longueur inférieure ou égale à k est alors $(1-2^{-k})^n$. Finalement on obtient que la probabilité que la taille M du plus grand préfixe de zéros soit égale à k, noté $\mathbb{P}_n(M=k)$, vérifie

$$\mathbb{P}_n(M=k) = (1-2^{-k})^n - (1-2^{-k+1})^n. \tag{4.1}$$

Espérance de M

Pour calculer l'espérance de M, on considère la courbe déterminée par $\mathbb{P}_n(M \geq k) = 1-(1-2^{-k})^n$ lorsque k varie. On remarque tout d'abord que $(1-2^{-k})^n \sim e^{-n2^{-k}}$, et on va en fait étudier

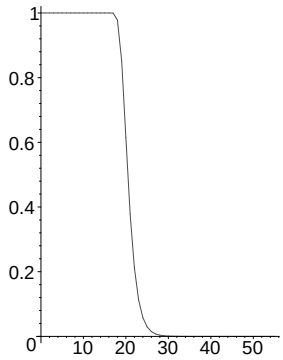

FIG. 4.8 – Tracé de la probabilité que $M \geq k$, pour $n = 10^6$ mots, pour $0 \leq k \leq 4 \log n$.

la courbe définie par $1 - e^{-n2^{-k}}$, qui est représentée Figure 4.8 pour $n = 10^6$. On observe sur cette courbe qu'il y a un effet de seuil, que si k est inférieur à $\log_2 n$ alors la probabilité est très proche de 1, et si k est supérieur à $\log_2 n$, elle vaut presque 0. On va maintenant prouver la réalité de ce phénomène, ce qui permettra ensuite d'évaluer l'espérance de M. Plus précisément, on énonce les propriétés suivantes :

- Si $k < \log_2 n - \log\log_2 n$. Alors $1 - e^{-n2^{-k}} > 1 - 1/n$.
- Si $k < \log_2 n + \log\log_2 n$. Alors $1 - e^{-n2^{-k}} < 1 - 1/\log_2 n$, avec de plus une décroissance exponentielle de la queue de distribution.

L'espérance de M s'exprime comme $\sum_k \mathbb{P}_n(M \geq k)$. D'après ce qui précède, cette somme est équivalente à $\log_2 n$:

$$\mathbb{E}_n(M) \sim \log_2 n.$$

L'espérance de M est obtenue de manière beaucoup plus précise grâce à la transformation de Mellin. On dispose alors un développement asymptotique de $\mathbb{E}(M)$ jusqu'à n'importe quel ordre. Cette étude a été rappelée dans la section 1.2.2, et peut être trouvée dans [FGD95]. Le développement obtenu dans la section 1.2.2 est

$$\mathbb{E}_n(M) = \log_2 n + \frac{\gamma}{\log 2} + \epsilon(\log_2 n) + o(1).$$

où $\epsilon(n)$ est une fonction périodique de période 1, et d'amplitude bornée par 10^{-6}.

Choix de l'estimée

Sachant que l'espérance de M est équivalente à $\log_2 n$, il pourrait être tentant de considérer 2^M comme estimée possible de n. En fait l'espérance de 2^M est infinie.

L'idée suivante consiste à séparer les éléments en m groupes, qu'on appelle urnes, où m est un paramètre de contrôle de l'algorithme. Une méthode très simple pour réaliser cette séparation est de choisir $m = 2^k$, et d'utiliser les k premiers bits de chaque élément comme numéro de l'urne dans laquelle ils sont placés. Ensuite, pour chaque urne, on efface les k premiers bits de chaque mot (cette information a déjà été utilisée), et on calcule la valeur M du maximum des positions du premier 1 prise sur les éléments de cette urne. On obtient alors m maximums qu'on note M_0, ..., M_{m-1} (à l'urne numéro i correspond le maximum M_i). On s'intéresse alors à la moyenne arithmétique des valeurs M_i, $\frac{1}{m}\sum M_i$ comme estimée potentielle de $\log_2 n$. On verra plus tard, au cours de l'analyse, que l'espérance de la puissance de cette moyenne $m2^{\frac{1}{m}\sum M_i}$, est $\frac{n}{\alpha_m}$ (il y a une constante

de correction ,à des fluctuations très petites près, qui sont étudiées au cours de l'analyse). On pose donc comme estimée

$$E_n = \alpha_m 2^{\frac{1}{m}\sum M_i}. \tag{4.2}$$

Ensuite se pose la question de la qualité de la réponse, laquelle est déterminée par la variance de E_n. La définition de l'erreur standard permet de formaliser cette notion :

Définition 4.1 *Soit une variable aléatoire X de moyenne μ et de variance σ^2, l'erreur standard notée err est définie par*

$$err = \frac{\sqrt{\sigma^2 - \mu^2}}{\mu}$$

4.4.3 L'algorithme

Cette section présente formellement l'algorithme et les résultats associés, qui sont ensuite prouvés dans la section 4.5.

L'algorithme Loglog dans sa version de base prend en entrée un multiensemble $\mathfrak{M}$ de mots binaires infinis. La position du premier 1 dans le mot y en comptant à partir de 1 est notée $\rho(y)$. C'est-à-dire $\rho(1011...) = 1$ et $\rho(0100101...) = 2$.

Algorithme LogLog($\mathfrak{M}$;$m := 2^k$)
Initialiser $M_0, \ldots, M_{m-1}$ à 0 ;
soit $\rho(y)$ la position maximale du premier 1 dans y ;
    pour $x = b_1 b_2 \cdots \in \mathfrak{M}$ faire
        $j := \langle b_1 \cdots b_k \rangle_2$ (Valeur des k premiers bits en base 2)
        $M_j := \max(M_j, \rho(b_{k+1} b_{k+2} \cdots))$;
renvoyer $E_n := \alpha_m m 2^{\frac{1}{m}\sum_j M_j}$ comme estimée du cardinal.

On définit :

$$\alpha_m := \left(\Gamma(-1/m) \frac{1 - 2^{1/m}}{\log 2} \right)^{-m}, \qquad \Gamma(s) := \frac{1}{s} \int_0^\infty e^{-t} t^s \, dt. \tag{4.3}$$

Les valeurs numériques de différents α_m sont

$$\alpha_{64} \doteq 0.39178 \quad \alpha_{128} \doteq 0.39439 \quad \alpha_{1024} \doteq 0.39668 \quad \alpha_\infty \doteq 0.39701.$$

Le nom Loglog. L'algorithme stocke en mémoire la position du premier 1 pour chaque groupe. Ces valeurs sont situées en moyenne près de $\log_2 n$, et l'espace mémoire pour stocker chacun d'entre eux est donc proche de $\log_2 \log_2 n$, d'où le nom Loglog. Cette occupation mémoire de $m \log_2 \log_2 n$ est à comparer avec l'espace utilisé par la plupart des autres algorithmes qui est de $m \log_2 n$.
Remarque. Si on considère l'arbre digital associé aux mots de cet ensemble, l'algorithme détermine la taille de la branche gauche dans le cas $m = 1$, et la moyenne arithmétique de la taille des branches gauches commençant au niveau $\log_2 m$ dans le cas général.

donnée w	$(M_i, \rho(\text{suffixe}))$	$[M_0, M_1, M_2, M_3]$
1101111...	(3,2)	[0,0,0,2]
1111001...	(3,1)	[0,0,0,2]
1011011...	(2,1)	[0,0,1,2]
0100110...	(1,3)	[0,3,1,2]
0010101...	(0,1)	[1,3,1,2]
1111101...	(3,1)	[1,3,1,2]
0110110...	(1,1)	[1,3,1,2]
1100101...	(3,3)	[1,3,1,3]
1000011...	(2,4)	[1,3,4,3]
0101100...	(1,2)	[1,3,4,3]
0001010...	(0,2)	[2,3,4,3]
1001000...	(2,2)	[2,3,4,3]

FIG. 4.9 – Un exemple d'exécution de l'algorithme Loglog

Un exemple La figure 4.9 montre le déroulement de l'algorithme sur la base de données {0001010, 0010101, 0100110, 0101100, 0110110, 1000011, 1001000, 1011011, 1100101, 1101111, 1111001, 111110}, dans le cas où $m = 4$. Les registres sont désignés par M_0, M_1, M_2, M_3. La colonne de gauche présente les mots de l'ensemble étudié, la colonne du centre le couple constitué par le registre associé à la donnée et la position du premier 1 dans le suffixe et la colonne de droite l'évolution des registres.

Sachant que $\alpha_4 \approx 0.35$, le résultat renvoyé par l'algorithme pour cet exemple est ≈ 11.4.

4.4.4 Résultats

Théorème 4.1 *Lorsque l'algorithme de base* LOGLOG *est appliqué à un multiensemble idéal de cardinalité n, et si on note E_n la valeur retournée par l'algorithme alors*

(i) L'estimateur E_n est asymptotiquement non biaisée, c'est-à-dire que, lorsque $n \to \infty$,

$$\frac{1}{n}\mathbb{E}(E_n) = 1 + \theta_{1,n} + o(1), \qquad \text{où} \quad |\theta_{1,n}| < 10^{-6}.$$

(ii) L'erreur standard, définie comme $\frac{1}{n}\sqrt{\mathbb{V}(E_n)}$ vérifie quand $n \to \infty$,

$$\frac{1}{n}\sqrt{\mathbb{V}(E_n)} = \frac{\beta_m}{\sqrt{m}} + \theta_{2,n} + o(1), \qquad \text{où} \quad |\theta_{2,n}| < 10^{-6}.$$

Avec :

$$\beta_m = m\sqrt{\left(\Gamma(-2/m)\frac{1-2^{2/m}}{\log 2}\right)^m - \left(\Gamma(-1/m)\frac{1-2^{1/m}}{\log 2}\right)^{2m}},$$

et $\beta_{128} \doteq 1.30540, \quad \beta_{1024} \doteq 1.29897, \quad \beta_\infty = \sqrt{\frac{1}{12}\log^2 2 + \frac{1}{6}\pi^2} \doteq 1.29806.$

4.5 Analyse de l'algorithme

4.5.1 Résumé

La figure 4.10 résume le schéma suivi pour l'analyse de l'algorithme. On part d'une quantité qu'on veut étudier qui est notée f_n sur le schéma. Cette quantité est typiquement l'espérance de

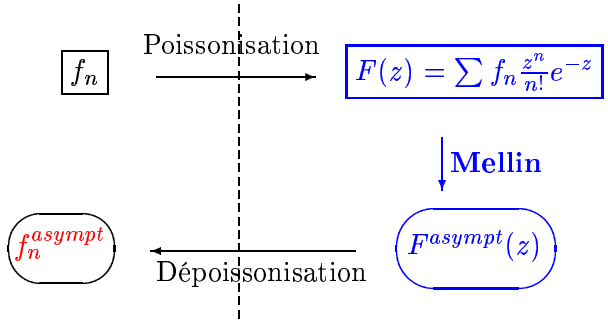

FIG. 4.10 – Schéma de l'analyse. A gauche des pointillés le monde Bernoulli, et à droite le monde Poisson.

l'estimateur ou sa variance. L'étude se poursuit dans le "monde" Poisson, qui correspond au fait de considérer des ensembles non pas de taille n fixée, mais des ensembles dont la taille suit une loi de Poisson de paramètre z. La série génératrice de Poisson $F(z)$ de ce paramètre peut ici être calculée de façon explicite.

La transformation de Mellin permet de calculer un équivalent asymptotique de la valeur du paramètre lorsque z tend vers l'infini, et enfin un théorème de dépoissonisation permet de revenir au modèle initial et d'obtenir un équivalent asymptotique du paramètre f_n, avec pour résultat $f_n \sim F(n)$.

Cette étude est traitée selon le plan suivant. La section 4.5.2 donne la mise en équation des séries génératrices, la section 4.5.3 détaille le sens de la poissonisation des séries génératrices obtenues. La section suivante 4.5.4 exprime l'asymptotique du paramètre sous le modèle Poisson par la transformation de Mellin, et enfin la section 4.5.5 dépoissonnise ces résultats et conclut.

4.5.2 Séries génératrices

On commence par modéliser le problème d'une seule urne. Lorsqu'une urne reçoit ν éléments, la variable aléatoire M associée (qui compte la position maximale du premier 1) a une distribution de probabilité qui est

$$\Pr_{\nu}(M \leq k) = \left(1 - \frac{1}{2^k}\right)^{\nu}, \qquad \Pr_{\nu}(M = k) = \left(1 - \frac{1}{2^k}\right)^{\nu} - \left(1 - \frac{1}{2^{k-1}}\right)^{\nu}.$$

La série génératrice bivariée exponentielle de cette distribution de probabilité est alors

$$G(z,u) := \sum_{\nu,k} \Pr_{\nu}(M = k) u^k \frac{z^\nu}{\nu!}. = \sum_k u^k \left(e^{z(1-1/2^k)} - e^{z(1-1/2^{k-1})}\right), \tag{4.4}$$

qu'on obtient par simple sommation. Cette série génératrice G permet d'exprimer la moyenne et la variance de la pseudo-estimée

$$Z := E_n/\alpha_m \equiv m 2^{\frac{1}{m}\sum_j M_j},$$

qui est la version non normalisée de l'estimée E_n.

On énonce le lemme qui exprime la moyenne et la variance de Z en terme de G.

Lemme 4.2 *La moyenne et la variance de l'estimée non normalisée Z valent*

$$\begin{aligned} \mathbb{E}(Z) &= mn![z^n]G\left(\tfrac{z}{m}, 2^{1/m}\right)^m \\ \mathbb{V}(Z) &= m^2 n![z^n]\left(G\left(\tfrac{z}{m}, 2^{2/m}\right)\right)^m - \left(m^n![z^n]G\left(\tfrac{z}{m}, 2^{1/m}\right)^m\right)^2. \end{aligned}$$

Preuve. La série génératrice G code le comportement d'une urne dans laquelle tombent ν éléments. Le fait d'élever cette série à la puissance m correspond à coder le comportement de m urnes dans lesquelles n éléments se répartissent de façon équiprobable (on doit remplacer z par z/m pour obtenir la normalisation correcte). Le coefficient $n![z^n]G(z/m,u)^m$ est alors la série génératrice de probabilité de la somme $\sum_j M_j$. Les expressions des premier et deuxième moments sont finalement obtenus en substituant $u \mapsto 2^{1/m}$ et $u \mapsto 2^{2/m}$. ■

La preuve du théorème 4.1 se réduit alors au fait d'estimer asymptotiquement ces deux moments.

4.5.3 Poissonisation

Dans cette sous-section, on "poissonnise" le problème d'estimation de la moyenne et de la variance. Une loi de Poisson de paramètre λ est la loi d'une variable aléatoire X telle que

$$\Pr(X = \ell) = e^{-\lambda}\frac{\lambda^\ell}{\ell!}.$$

Etant donné un ensemble de lois de probabilité M_λ, indexées par une valeur λ, la "poissonisation" consiste à considérer la nouvelle famille M_{X_λ}, où X_λ est une loi de Poisson de paramètre λ. Autrement dit, pour le problème qui nous intéresse, au lieu de jeter λ éléments dans les urnes, on en jette une quantité qui suit une loi de Poisson de paramètre λ. L'intérêt de ce changement de modèle est qu'on peut utiliser la transformation de Mellin, laquelle permet d'estimer asymptotiquement l'espérance et la variance de manière commode. La sous-section suivante est consacrée à la dépoissonisation de ces résultats. L'idée est intuitivement justifiée par le fait que la loi de Poisson est très concentrée autour de sa moyenne, et donc que jeter n éléments ou une quantité choisie selon une loi de Poisson de paramètre λ, pour λ égal ou voisin de n, conduit à des phénomènes quantitativement semblables.

L'intérêt du modèle Poisson est que les moments se calculent sans problème. Tout d'abord, l'espérance et la variance s'expriment grâce à la série génératrice bivariée de probabilité G introduite pour le modèle non-poisson. Ceci est une règle générale que si $f(z) = \sum_n f_n z^n/n!$ est la série génératrice exponentielle qui code l'espérance d'un paramètre, alors

$$e^{-\lambda} f(\lambda) = \sum_n f_n e^{-\lambda}\frac{\lambda^n}{n!}$$

est l'espérance sous le modèle Poisson correspondant.

Une conséquence directe de cette règle est que les quantités

$$\begin{cases} \mathcal{E}_n &= mG\left(\frac{n}{m}, 2^{1/m}\right)^m e^{-n} \\ \mathcal{V}_n &= m^2 G\left(\frac{n}{m}, 2^{2/m}\right)^m e^{-n} - \left(mG\left(\frac{n}{m}, 2^{1/m}\right)^m e^{-n}\right)^2. \end{cases} \tag{4.5}$$

sont respectivement la moyenne et la variance de Z lorsque le nombre d'éléments distincts de l'ensemble étudié suit une loi de Poisson de paramètre n.

4.5.4 Mellin

L'asymptotique de la moyenne et de la variance de l'estimée sous le modèle Poisson est obtenue par la transformation de Mellin :

Lemme 4.3 *La moyenne et la variance sous le modèle Poisson $\mathcal{E}_n$ et $\mathcal{V}_n$ vérifient quand $n \to \infty$:*

$$\begin{aligned} \mathcal{E}_n &\sim \left[\left(\Gamma(-1/m)\tfrac{1-2^{1/m}}{\log 2}\right)^m + \epsilon_n\right] \cdot n \\ \mathcal{V}_n &\sim \left[\left(\Gamma(-2/m)\tfrac{1-2^{2/m}}{\log 2}\right)^m - \left(\Gamma(-1/m)\tfrac{1-2^{1/m}}{\log 2}\right)^{2m} + \eta_n\right] \cdot n^2. \end{aligned} \tag{4.6}$$

où $|\epsilon_n|$ et $|\eta_n|$ sont bornés par 10^{-6}.

La preuve est essentiellement basée sur la transformation de Mellin, présentée en introduction dans la section 1.2. Par commodité, on rappelle ici très brièvement sa définition ainsi que les propriétés essentielles qui sont utilisées pour cette preuve. La transformée de Mellin d'une fonction f définie sur l'axe réel est définie par

$$f^*(s) = \int_0^\infty f(t) t^{s-1}\, dt.$$

Les deux principales propriétés de la transformation de Mellin utilisées ici sont :

1. Il y a une correspondance entre les propriétés asymptotiques de la fonction f et les singularités de la transformée de Mellin f^*.
2. Les sommes harmoniques de la forme $\sum \lambda_k f(\mu_k x)$ ont une transformée de Mellin qui se factorise sous la forme $\left(\sum \lambda_k \mu_k^{-s}\right) \cdot f^*(s)$.

Preuve. L'espérance et la variance $\mathcal{E}_n$ et $\mathcal{V}_n$ définies dans équation (4.5) peuvent être récrites comme

$$\mathcal{E}_n = mA(n)^m, \qquad \mathcal{V}_n + \mathcal{E}_n^2 = m^2 B(n)^m,$$

où $A(x)$ et $B(x)$ désignent les sommes harmoniques

$$A(x) = \sum_i 2^{i/m} \left(\varphi(x/2^i) - \varphi(x/2^{i-1})\right), \quad B(x) = \sum_i 2^{2i/m} \left(\varphi(x/2^i) - \varphi(x/2^{i-1})\right),$$

avec $\varphi(x) = e^{-x/m}$. Ceci découle de l'expression de G dans l'équation (4.4).

La transformée de Mellin de φ est $\varphi^*(s) = m^s\Gamma(s)$, et, par application de la transformation de Mellin aux sommes harmoniques (cf [FGD95]) on obtient

$$A^*(s) = \varphi^*(s)(2^s - 1)\frac{2^{1/m}}{1 - 2^{1/m}2^s}, \quad B^*(s) = \varphi^*(s)(2^s - 1)\frac{2^{2/m}}{1 - 2^{2/m}2^s}.$$

Espérance

La bande fondamentale de $A^*(s)$ est $\langle -1, -1/m\rangle$. La figure 4.11 montre la bande fondamentale de $A^*(s)$ (en hachuré) et ses singularités. Les singularités de $\frac{1}{1-2^{1/m}2^s}$ sont désignées par des cercles, et celles de $\Gamma(s)$ par des carrés. Les singularités de $A^*(s)$ sur l'axe vertical $\Re(s) = -1/m$ sont situées aux points $\Im(s) \equiv 0 \mod (2\pi/\log 2)$. Chacune de ces singularités fournit une contribution au développement asymptotique de $A(x)$. La contribution principale provient de la singularité située sur l'axe réel. Au voisinage de $s = -1/m$, on a

$$A^*(s) \sim \frac{-a}{s + 1/m}, \qquad \text{avec} \quad a = m^{-1/m}\Gamma(-1/m)\frac{1 - 2^{1/m}}{\log 2}. \tag{4.7}$$

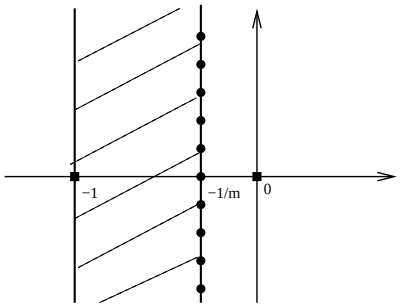

FIG. 4.11 – Les singularités et la bande fondamentale de $A^*(s)$.

Le théorème de transfert de Mellin nous dit que la contribution correspondant à cette singularité dans le développement asymptotique de $A(x)$ à l'infini est $ax^{1/m}$. Pour les singularités complexes situées aux points $-1/m + 2ik\pi/\log 2$, le développement au voisinage de ces singularités est

$$A^*(s) \sim \frac{-a_k}{s+1/m}, \qquad \text{avec} \quad a_k = m^{-1/m}\Gamma\left(-1/m + \frac{2ik\pi}{\log 2}\right)\frac{1-2^{1/m}}{\log 2}.$$

Chacune de ces singularités ajoute donc une contribution au développement asymptotique de $A(x)$ qui est de la forme $a_k x^{1/m+2ik\pi/\log 2}$. La décroissance très rapide de la fonction Γ le long d'un axe vertical entraîne que ces constantes a_k sont extrêmement petites. Par exemple $|\Gamma(2i\pi/\log 2)| \approx 5 \cdot 10^{-7}$ et $|\Gamma(4i\pi/\log 2)| \approx 3 \cdot 10^{-12}$. Par conséquent, la valeur $\mathcal{E}_n = mA(n)^m$ est équivalente à $ma^m n + \epsilon_n n$ quand n tend vers l'infini, où ϵ_n est une petite fluctuation. Ceci prouve l'équivalent de la moyenne énoncée dans l'équation (4.6).

Remarque. Les fluctuations de faibles amplitudes sont inhérentes au problème, et rendent de fait l'analyse intrinsèquement non-élémentaire. En pratique ces fluctuations sont indétectables car masquées par les variations statistiques.

Variance

Pour la variance, on montre que la bande fondamentale de $B^*(s)$ est $\langle -1, -2/m\rangle$. Les pôles qui contribuent au premier ordre du développement asymptotique de $B(x)$ à l'infini sont donc situés sur l'axe de partie réelle $-2/m$. La contribution principale est donnée par la singularité $s = -2/m$, au voisinage de laquelle on a

$$B^*(s) \sim b\frac{-1}{s+2/m} \qquad \text{avec} \quad b = m^{-2/m}\Gamma(-2/m)\frac{1-2^{2/m}}{\log 2}. \tag{4.8}$$

La contribution de cette singularité est donc $bx^{2/m}$. De façon similaire à l'espérance, les autres singularités contribuent sous la forme de fluctuations de faible amplitude. On a donc prouvé que $B(x) \approx bx^{2/m} + \delta_x x^{2/m}$, où δ_x est périodique, et d'amplitude très faible. L'estimation de la variance énoncée dans l'équation (4.6) s'ensuit. ■

4.5.5 Dépoissonisation

Les estimations asymptotiques de la moyenne et de la variance de l'estimée obtenues pour le modèle de Poisson peuvent être transférées au modèle initial grâce à un théorème de Dépoissonisation.

Plusieurs cadres sont présentés dans le chapitre 10 du livre de Szpankowski [Szp01]. On utilise ici la méthode nommée "dépoissonisation analytique" due à Jacquet et Szpankowski, laquelle se base sur une analyse de col appliquée aux intégrales de Cauchy, cf [JS98, Szp01].

Lemme 4.4 *Les deux premiers moments de l'estimée de l'algorithme* LogLog *sont asymptotiquement équivalent, dans le modèle de Poisson et sous le modèle qui fixe la valeur de n :*

$$\mathbb{E}_n(Z) \sim \mathcal{E}_n, \qquad \mathbb{V}_n(Z) \sim \mathcal{V}_n$$

Preuve.

Tout d'abord on définit le cône S_θ comme

$$S_\theta = \{z : |\arg\ z| \leq \theta\},\ \text{avec}\ |\theta| < \pi/2.$$

Le "lemme de dépoissonisation basique" de [JS98] instancié à notre problème se reformule comme suit

Lemme 4.5 *Supposons qu'il existe $\theta > 0$, $\alpha < 1$, tels que*
(C_1) : à l'intérieur de S_θ on a $e^{-z}G\left(z/m, 2^{1/m}\right)^m = O(|z|)$,
(C_2) : à l'extérieur du cône S_θ, on a $G\left(z/m, 2^{1/m}\right)^m = O(e^{\alpha|z|})$.
Alors $\mathbb{E}(Z_n) \sim \mathcal{E}_n$.

Pour vérifier les hypothèses de ce lemme de dépoissonisation, on récrit la série génératrice $G(z,u)$ définie dans l'équation (4.4) :

$$G(z, 2^{1/m}) = e^z \sum_k 2^{k/m} \left(e^{-\frac{z}{2^k}} - e^{-\frac{z}{2^{k-1}}}\right).$$

Condition C_2. On étudie séparément les deux zones $\Re(z) < 0$, et $\Re(z) \geq 0$.

• Pour la zone $\Re(z) < 0$, on obtient $G(z, 2^{1/m}) = O(e^{\alpha|z|})$ en séparant la somme en deux parties selon la valeur $L := \lfloor \log_2 |z| \rfloor$, soit $k \leq L$, et $k > L$. Pour la deuxième partie, on utilise l'égalité $\left(e^{-\frac{z}{2^k}} - e^{-\frac{z}{2^{k-1}}}\right) = O(z/2^k)$, ce qui fournit

$$G(z, 2^{1/m}) = O\left(e^z \sum_{k \leq L} 2^{L/m} 2e^{-z}\right) + O\left(e^z \sum_{k>L} 2^{k/m} \frac{z}{2^k}\right).$$

On a ainsi obtenu que $G(z, 2^{1/m}) = O\left(e^{\alpha|z|}\right)$, quel que soit $\alpha > 0$.

• Lorsque $\Re(z) \geq 0$, on a encore $\left(e^{-\frac{z}{2^k}} - e^{-\frac{z}{2^{k-1}}}\right) = O(z/2^k)$, ce qui permet d'écrire que

$$G(z, 2^{1/m}) = O\left(e^z \sum_k 2^{k/m} \frac{z}{2^k}\right) = O(ze^z).$$

Si on choisit $\theta = \pi/3$, alors en prenant $\alpha = 3/4$, on a $G(z, 2^{1/m}) = O(e^{\alpha|z|})$.

Le résultat $G(z, 2^{1/m}) = O(e^{\alpha|z|})$ entraîne que $G(z/m, 2^{1/m})^m = O(e^{\alpha|z|})$, ce qui est la condition C_2.

Condition C_1. La condition C_1 est démontrée grâce à la transformation de Mellin. Pour la cas où z est réel positif, on a vu dans la section 4.5 que $e^{-z}G(z/m, 2^{1/m})^m \sim z$. Ceci implique la condition C_1. Lorsque z est situé à l'intérieur du cône S_θ, mais n'est pas réel, les mêmes résultats peuvent être obtenus en adaptant légèrement la preuve.

Posons $A_\tau(x) = G(xe^{i\tau}/m, 2^{1/m})e^{-xe^{i\tau}/m}$, où $|\tau| \leq \pi/3$, ce qui permet d'étudier la fonction G selon une direction complexe fixée pour z. La fonction A_τ définie ici est égale à la fonction A définie dans la section 4.5.4 lorsque l'angle τ est nul, de plus on peut exprimer A_τ en fonction de A par $A_\tau(x) = A(xe^{i\tau})$. On reprend les grandes étapes de la démonstration du lemme 4.2 à propos de l'espérance.

La fonction A_τ s'écrit sous forme de somme harmonique

$$\begin{aligned} A_\tau(x) &= \sum_k 2^{k/m} \left(e^{-\frac{xe^{i\tau}}{m2^k}} - e^{-\frac{xe^{i\tau}}{m2^{k-1}}} \right) \\ &= \sum_k 2^{k/m} \left(\Psi\left(\frac{x}{2^k}\right) - \Psi\left(\frac{x}{2^{k-1}}\right) \right). \end{aligned}$$

où Ψ est définie par $\Psi(x) = e^{-\frac{xe^{i\tau}}{m}}$. D'après ce qu'on sait des transformées de séries harmoniques, on obtient que la transformée de Mellin de A_τ est

$$A_\tau^*(s) = \Psi^*(s)(2^s - 1)\frac{2^{1/m}}{1 - 2^{1/m}2^s}.$$

La transformée de Mellin de Ψ s'obtient de la façon suivante

$$\Psi^*(s) = \int_0^\infty e^{-\frac{xe^{i\tau}}{m}} x^{s-1} dx = \int_0^{e^{i\tau}\infty} e^{-x} x^{s-1} (e^{i\tau}/m)^{-s} dx.$$

Pour calculer cette intégrale, on décompose le chemin d'intégration en un chemin qui suit l'axe réel, et un chemin qui va de l'axe réel à l'axe complexe d'angle τ. Formellement, on écrit $\int_0^{Re^{i\tau}} = \int_0^R + \int_R^{Re^{i\tau}}$, où la troisième intégrale est le long du cercle de rayon R. Ensuite on passe à la limite en faisant tendre R vers l'infini. La première intégrale tend alors vers $\Psi^*(s)$, la deuxième vers $(e^{i\tau}/m)^{-s} \int_0^\infty e^{-x} x^{s-1} dx = (e^{i\tau}/m)^{-s}\Gamma(s)$, et enfin la troisième intégrale qui est bornée par $R\tau e^{-R} R^{s-1} K$ (ou K est une constante) tend vers 0. On a finalement $\Psi^*(s) = \left(\frac{e^{i\tau}}{m}\right)^{-s} \Gamma(s)$.

La transformée de Mellin de A_τ peut être écrite de façon complète

$$A_\tau^*(s) = \left(\frac{e^{i\tau}}{m}\right)^{-s} \Gamma(s)(2^s - 1)\frac{2^{1/m}}{1 - 2^{1/m}2^s},$$

dont la bande fondamentale est $\langle -1, -1/m \rangle$. Les singularités de $A_\tau^*(s)$ à droite de la bande fondamentale sont situées le long de l'axe $\Re(s) = -1/m$. Le théorème 1.2 s'applique alors à condition que l'hypothèse de décroissance $f^*(s) = O(|s|^{-r})$, $r > 1$, soit satisfaite. On connaît [AS73, p257] une majoration pour la décroissance de la fonction Γ à l'infini qui est

$$\Gamma(x + iy) = O(e^{-|y|\pi/2.1}).$$

Cette décroissance va compenser la croissance du terme $e^{-is\tau}$ dans le cas qui nous intéresse. On note $s = c + id$, ce qui permet d'écrire que $e^{-is\tau} = O(e^{d\pi/3})$. Lorsqu'on injecte ces bornes dans l'expression de $A_\tau^*(s)$, le résultat est $A_\tau^*(s) = O(e^{-|d|\pi 0.1}) = O(|s|^{-2})$. Le théorème 1.2 s'applique, et

dit que l'asymptotique de $A_\tau(x)$ est en $O(x^{1/m})$ lorsque x tend vers l'infini. Ce résultat est valable pour tout τ inférieur à $\theta = \pi/3$ l'angle du cône, ce qui prouve la condition C_1 car cette condition est équivalente à $A_\tau(x)^m = O(x)$ quel que soit $|\tau| \leq \pi/3$, et pour x grand.

Les deux conditions C_1 et C_2 étant satisfaites, le lemme 4.5 s'applique, ce qui achève la preuve du lemme 4.4.

La preuve pour la variance est similaire, et n'est pas détaillée.

■

On peut maintenant terminer la preuve du théorème 4.1. L'estimateur non normalisé Z, associé à la cardinalité sous-jacente n est asymptotiquement équivalent à n/α_m par les lemmes 4.3 et 4.4. L'estimateur $E_n = \alpha_m Z$ est donc asymptotiquement non biaisé. La variance, si on néglige les fluctuations mineures, est asymptotiquement équivalente à $\sqrt{b^m a^{-2m} - 1}$, où a, b sont définies dans les équations (4.7) et (4.8). Grâce encore aux lemmes 4.3 et 4.4, cette valeur est précisément équivalente à $\beta_m/\sqrt{m}$ selon les notations du théorème 4.1.

Chapitre 5

Super Loglog et expérimentation

Sommaire

Ce chapitre regroupe des résultats expérimentaux obtenus avec l'algorithme Loglog, ainsi que des optimisations suggérées par ces expériences. Les améliorations de l'algorithme Loglog, notamment la variante super Loglog, se fondent sur une analyse basée sur quelques hypothèses très réalistes. Les effets de ces optimisations sont ensuite examinés dans la section 5.6 entièrement dédiée aux résultats expérimentaux. Ces expériences sont bien sûr effectuées sur des mots de taille finie, et la pertinence du modèle des mots infinis est justifiée en détail dans la section 5.4.

Résultats principaux. Ce chapitre présente deux algorithmes d'estimation de cardinalité d'ensembles de données réels, Loglog et super Loglog. Ces algorithmes sont les meilleurs disponibles actuellement. Ils sont non biaisés même sur des ensembles de petite cardinalité.

La section 5.6 fournit de plus une validation expérimentales de tous les résultats annoncés.

5.1 Introduction

L'algorithme Loglog

On rappelle que l'algorithme Loglog permet d'estimer la cardinalité d'un ensemble de données, en utilisant une mémoire auxiliaire très restreinte.

Cet algorithme a été étudié dans le chapitre 4, et est rappelé ici. L'algorithme Loglog de base prend en entrée un multiensemble $\mathfrak{M}$ de mots binaires infinis. La position du premier 1 dans un mot y (en comptant à partir de 1) est notée $\rho(y)$. Par exemple $\rho(1011...) = 1$ et $\rho(0100101...) = 2$.

Algorithme LogLog($\mathfrak{M}$; $m := 2^k$)
Initialiser $M_0, \ldots, M_{m-1}$ à 0 ;
soit $\rho(y)$ la position maximale du premier 1 dans y ;
pour $x = b_1 b_2 \cdots \in \mathfrak{M}$ faire
$j := \langle b_1 \cdots b_k \rangle_2$ (Valeur des k premiers bits en base 2)
$M_j := \max(M_j, \rho(b_{k+1} b_{k+2} \cdots)$;
renvoyer $E_n := \alpha_m m 2^{\frac{1}{m}\sum_j M_j}$ comme estimée du cardinal

Les quantités α_m sont définies par

$$\alpha_m := \left(\Gamma(-1/m) \frac{1 - 2^{1/m}}{\log 2} \right)^{-m}, \qquad \Gamma(s) := \frac{1}{s} \int_0^\infty e^{-t} t^s \, dt.$$

Super Loglog

L'algorithme Loglog est très simple à énoncer et à programmer. Les résultats expérimentaux permettent de vérifier les prédictions fournies par l'analyse et développées dans le chapitre précédent. Par exemple les résultats de la section 4.4.4 montrent que l'estimateur est non biaisé et que l'erreur standard est voisine de $1.3/\sqrt{m}$. La figure 5.10 confirme fort bien le comportement en $1.3/\sqrt{m}$ de l'erreur standard.

L'analyse du chapitre précédent, section 4.5, permet de comprendre les phénomènes associés à des paramètres éventuellement difficiles à observer expérimentalement, par exemple de très petites oscillations ou des distributions limites. Cette compréhension fine guide ensuite la mise au point de l'algorithme, et dans le cas présent la conception d'une variante plus efficace, le super Loglog. Ce nouvel algorithme est inspiré par l'étude de la distribution des valeurs des registres pour l'algorithme Loglog, et sa mise au point est fortement dépendante de son analyse, qui est effectuée dans la section 5.8

L'algorithme Loglog, décrit dans le chapitre précédent est basé sur m estimations séparées de $\log_2 n/m$, où n désigne toujours le nombre d'éléments distincts que l'on souhaite estimer. Ces estimations étant obtenues en distribuant les éléments en m groupes selon leur préfixe. Chaque groupe donne ensuite lieu à une estimation de $\log_2 n/m$. Ces m estimations sont alors utilisées de la façon la plus simple possible dans l'algorithme Loglog, car on calcule la moyenne arithmétique de toutes ces valeurs afin d'obtenir une meilleure estimation de $\log_2 n/m$. On peut bien sûr imaginer d'autres procédés pour extraire l'information contenue dans ces m valeurs. La variante *super Loglog* présentée dans la section 5.2 propose un usage de ces valeurs qui reste très simple et permet de gagner en précision. L'idée est de ne prendre en compte qu'une fraction des valeurs obtenues, les plus petites, et de prendre la moyenne arithmétique de ces plus petites valeurs comme estimation de $\log_2 n/m$.

La section 5.8 est dédiée à l'analyse de ce nouvel algorithme super Loglog. L'espérance et la variance de l'estimation renvoyée par cet algorithme sont obtenues numériquement. Cette analyse permet de mettre à jour un phénomène d'oscillation de l'estimation due à la troncature des registres. L'étude de ces oscillations périodiques permet de mettre au point de façon simple l'algorithme super Loglog, et d'obtenir un nouvel algorithme non biaisé (i.e. dont l'espérance est le nombre de mots distincts). Cet algorithme aurait été très délicat à mettre en place en se basant uniquement sur des données expérimentales. Cette variante de l'algorithme Loglog reste extrêmement simple à implanter, et donne lieu à une erreur standard inférieure à $0.95/\sqrt{m}$. Les énoncés obtenus dans cette section sont basés sur quelques hypothèses simplificatrices. Pour ne pas introduire de confusion, on les nommera *Résultat*, et non pas théorème ou proposition.

Petits ensembles et données réelles.

La section suivante 5.3 étudie un travers de l'algorithme Loglog tel qu'il a été présenté pour l'instant. Les résultats sont entachés d'un biais systématique lorsque n est petit. L'estimation renvoyée par l'algorithme devient bonne dès que n est supérieur à environ $5m/2$. Ces non-linéarités initiales ont lieu tandis qu'on dispose d'une très bonne information, elles peuvent donc être corrigées sans problème, grâce à un algorithme de comptage de collisions. Le comportement de l'algorithme après cette modification devient alors excellent lorsque n est petit, et il suit ensuite les prévisions de précision annoncées par la théorie asymptotique.

La présentation de l'algorithme Loglog (ou super Loglog) est basée sur l'utilisation de mots infinis. En réalité l'algorithme s'applique à des mots finis (typiquement de longueur 32 ou 64 bits). Cela peut être modélisé par une troncature des mots infinis, ce qui est formalisé dans la section 5.4.1.

La section 5.4.2 est dédiée au problème des collisions lors du hachage des données. On dispose au départ d'un ensemble de données qui contient n éléments distincts, qu'on hache sur l'ensemble des mots binaires de taille L. On sait (paradoxe des anniversaires[Knu98, section 6.4]) que lorsque n est supérieur à $\sqrt{2^L}$, il est probable qu'il y ait des collisions. Autrement dit, que deux éléments distincts aient la même image par la fonction de hachage. Le nombre $\tilde{n}$ d'éléments distincts après hachage, qu'on estime par exemple par l'algorithme Loglog est donc a priori différent du nombre n qu'on recherche. En fait on sait très bien estimer le nombre de collisions (cf section 4.3), et en apportant les corrections de la section 4.3, on peut retrouver sans problème une estimation fiable de n à partir de l'estimation de $\tilde{n}$.

On montre dans la section 5.5 que la distribution des résultats de l'algorithme Loglog est asymptotiquement gaussienne, ce qui permet de mieux quantifier l'erreur commise.

Résultats expérimentaux

Les résultats expérimentaux sont présentés dans la section 5.6. Ces résultats concernent les algorithmes Loglog et super Loglog, avec ou sans les améliorations présentées précédemment. Les exécutions de ces algorithmes sont faites sur des données réelles diverses et variées (œuvres littéraires, traces de pages webs, décimales de π,...) ainsi que sur des données aléatoires.

La section 5.7 est consacrée à une amélioration finale de l'algorithme, basée sur la compression des registres. En effet, on a pour l'instant supposé que si les registres contenaient des valeurs inférieures à 32, ils étaient stockés sur 5 bits en utilisant un codage classique. Comme les valeurs de ces registres sont concentrées autour de $\log_2 n/m$, il est intéressant de compresser l'information contenue dans les registres. En utilisant un codage préfixe très simple, on réussit à n'utiliser environ que 3 bits (au lieu de 5) par registre, ce qui permet un gain de mémoire conséquent.

Finalement la dernière section 5.9 contient le code source commenté des algorithmes, en C.

5.2 Super Loglog

Cette section présente une variante de l'algorithme Loglog, nommée super Loglog. Ce nouvel algorithme a l'avantage d'être (à mémoire égale) plus précis que le précédent, d'environ 30%. Le traitement des données est identique, ce qui permet de conserver la simplicité et la rapidité de Loglog. Ce qui change est la manière dont sont utilisées les valeurs des registres M_i. Au lieu de calculer leur moyenne pour obtenir une estimation de $\log_2 n/m$, l'algorithme super Loglog ne conserve que la partie considérée comme la plus significative de ces valeurs, en l'occurrence les plus petites, et garde la moyenne arithmétique de ces valeurs sélectionnées comme estimation de $\log_2 n/m$. Une telle moyenne tronquée fournit de meilleurs résultats, car la distribution des registres a une queue

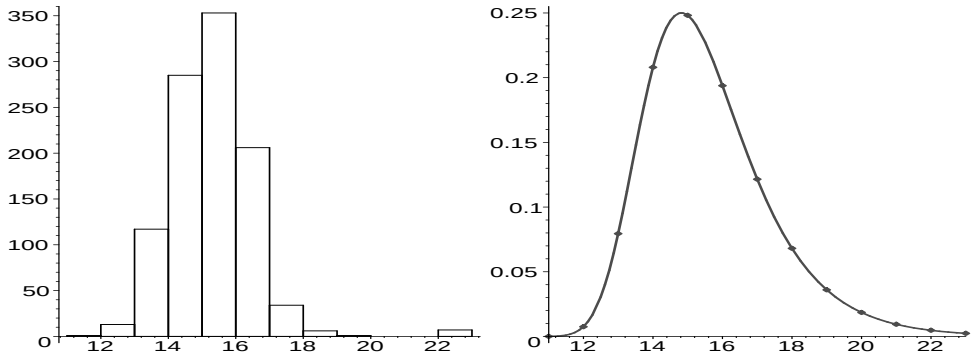

FIG. 5.1 – La distribution expérimentale des valeurs des maxima pour les digits de π, avec $n \approx 2 \cdot 10^7$ et $m = 1024$ [gauche] ; la distribution théorique de la valeur d'une urne M, si $\nu = 2 \cdot 10^4$, $\Pr_\nu(M = k)$[droite].

de distribution pour les grandes valeurs qui décroît relativement lentement (ce qui rend les grandes valeurs moins significatives).

Dans cette section on expose tout d'abord l'intérêt de ne prendre en compte qu'une partie des registres, puis en 5.2.2 on donne l'algorithme super Loglog en pseudo-code ; enfin les résultats principaux concernant cet algorithme sont énoncés dans le Résultat 5.1. L'analyse de cet algorithme est l'objet de la section 5.8.

5.2.1 Distribution des registres

Lors de l'exécution de l'algorithme Loglog, il y a $m = 2^k$ urnes qui contiennent chacune le maximum M d'un ensemble de variables suivant une loi géométrique. En première approximation, pour chaque urne, le nombre d'éléments pris en compte est le même, et est voisin de n/m. On peut donc étudier la distribution du maximum M, qui est très similaire pour tous les registres. La distribution de M est décrite comme suit, si l'urne reçoit n/m éléments :

$$\Pr_{n/m}(M = k) = (1 - 2^{-k})^{n/m} - (1 - 2^{-k+1})^{n/m}. \tag{5.1}$$

La figure 5.1 montre le tracé de la distribution de M, comparé à ce qu'on observe empiriquement sur une simulation. L'exemple choisi part des 200 millions premiers digits de π qu'on regroupe 10 par 10 de sorte à former 20 millions de mots. Cet exemple est réutilisé plus tard dans les études expérimentales.

On observe sur les tracés de la figure 5.1 que la courbe de distribution décroît assez lentement lorsque k devient grand. C'est ce qui explique, sur la courbe expérimentale la présence de valeurs de maxima isolés sur la droite (situés à la valeur $k = 22$ pour l'exemple de la figure 5.1). L'équation (5.1) précise que cette décroissance est géométrique, car lorsque k tend vers l'infini on a $\Pr_\nu(M = k) \sim \frac{\nu}{2^{k+1}}$. Ceci a pour conséquence une variabilité sur les grandes valeurs de M, lesquelles peuvent se trouver dans une zone trop grande et ainsi induire une erreur significative.

Une solution à ce problème consiste à tronquer les grandes valeurs des registres, en ne gardant qu'une fraction des plus petits (typiquement 70%), ce qui a pour effet de diminuer la variabilité de l'estimateur et donc d'améliorer la précision. Ce gain est cependant contrebalancé par le fait qu'on supprime des valeurs et qu'on ne prend donc en compte qu'une fraction des données dont on dispose. L'effet global résultant de ces deux phénomènes est étudié dans les sections suivantes.

5.2.2 Super Loglog, l'algorithme

Le nouvel algorithme avec troncature nommé *Super Loglog* est une variante de l'algorithme Loglog présenté dans la section 4.4. La différence entre les deux algorithmes est que, lors de l'estimation, une partie des valeurs des registres est éliminée et l'on ne garde que les plus petites. Le traitement des données et la mise à jour des registres sont les mêmes que pour l'algorithme Loglog, ce qui fait que l'espace mémoire mis en jeu et le temps de traitement sont identiques. Par contre, la fonction donnant l'estimateur est légèrement plus complexe. L'analyse montre que la "constante" de correction dépend en fait légèrement de la partie fractionnaire de $\log_2 n$. C'est une fonction périodique, de période $\log_2 n$, suffisamment régulière, et qui peut être approchée de façon précise par un polynôme, avec pour résultat que le calcul de l'estimateur reste rapide. L'algorithme calcule donc une première estimation temporaire n_0 par la méthode Loglog, puis grâce à cette estimation, peut déterminer avec une bonne précision un facteur de correction $\tilde{\alpha}_m(n_0)$ appliqué à l'estimateur basé sur la moyenne tronquée.

L'algorithme en pseudo-code est alors :

Algorithme SUPER LOGLOG($\mathfrak{M}$; $m := 2^k$)
Initialiser $M_0, \ldots, M_{m-1}$ à 0 ;
soit $\rho(y)$ la position maximale du premier 1 dans y ;
  for $x = b_1 b_2 \cdots \in \mathfrak{M}$ do
    $j := \langle b_1 \cdots b_k \rangle_2$ (Valeur des k premiers bits en base 2)
    $M_j := \max(M_j, \rho(b_{k+1} b_{k+2} \cdots))$;
Soit $\tilde{M}_1, \ldots, \tilde{M}_{\lfloor m\delta \rfloor}$ l'ensemble des $\lfloor m\delta \rfloor$ plus petites valeurs parmi les M_i ;
Soit $n_0 := \alpha_m m 2^{\frac{1}{m}\sum M_i}$;
$\kappa_{n_0} := \lceil \log_2(n_0/m) - \log_2 \log(1/\delta) \rceil - \log_2(n_0/m)$
renvoyer $\tilde{e}_n := P(\kappa_{n_0}) m 2^{\frac{1}{\lceil m\delta \rceil}\sum_j \tilde{M}_j}$ comme estimation du cardinal.

La fonction notée P est un polynôme qui approxime la fonction réelle de correction $\tilde{\alpha}_m(n_0)$ sur une période. Cette fonction $\tilde{\alpha}_m(n_0)$ fournit le facteur de correction du biais de l'algorithme. Elle dépend du taux de troncature δ, et est définie précisément dans la section analyse. Par exemple pour $\delta = 0.7$, on a $\tilde{\alpha}_\infty(n_0)$ qui oscille entre 0.764 et 0.782. La section 5.8.2 montre que ces oscillations qui dépendent de la valeur de n sont prises en compte par cette fonction. La valeur δ est un paramètre de réglage dont le choix est discuté dans la section analyse 5.8, et dans la partie résultats expérimentaux 5.6.

Par exemple, pour m grand et $\delta = 0.7$, le polynôme $P(z)$ est défini par

$$P(z) := 0.0030508z^4 - 0.034178z^3 + 0.19685z^2 - 0.47075z + 1.14288 \tag{5.2}$$

L'algorithme Loglog est modifié pour ne prendre en compte que la fraction δ des premières valeurs en compte. La valeur optimale de δ peut être estimée expérimentalement, en se basant sur des données aléatoires; d'après les résultats du tableau 5.3 qui repose sur les valeurs $m = 2^8$, $n = 40000$ et sur 10000 essais, on trouve que la valeur $\delta = 0.7$ est optimale.

δ	0.6	0.7	0.8	0.9
Erreur standard	7.3	6.5	7.1	7.3
Erreur standard $\times \sqrt{m}$	1.16	1.04	1.13	1.16

(5.3)

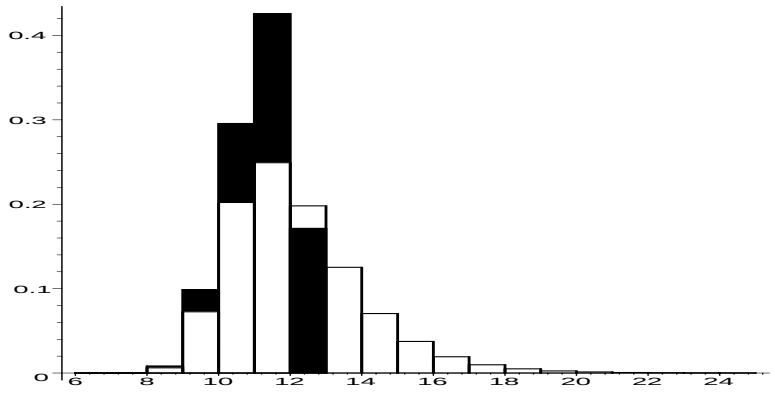

FIG. 5.2 – La distribution de probabilité avant (en blanc) et après troncature (en noir).

Pour le cas ou $\delta = 0.7$, les valeurs de la fonction $\tilde{\alpha}_m(n_0)$ varient comme suit :

	$\tilde{\alpha}_{128}(n_0)$	$\tilde{\alpha}_{256}(n_0)$	$\tilde{\alpha}_{1024}(n_0)$	$\tilde{\alpha}_{\infty}(n_0)$
Min	0.762	0.763	0.764	0.764
Max	0.780	0.7805	0.781	0.782

De façon surprenante, cette troncature va apporter un nouveau phénomène d'oscillations de l'estimateur selon les valeurs de $\log_2 n$. Ce phénomène est étudié dans la section 5.8.2. Il dépend de la manière dont se passe la troncature. On voit sur la figure 5.2 que la troncature se fait sur une distribution d'une variable aléatoire à valeurs entières. Selon que la coupure a lieu ou non entre deux valeurs, les phénomènes sont quelque peu différents. Une fois quantifiée, cette variation périodique de période $\log_2 n$ peut ensuite être corrigée.

Résultat 5.1 *L'algorithme Super Loglog qui estime le cardinal d'un ensemble, avec le choix $\delta = 0.7$, grâce à la formule*

$$\tilde{e}_n := \tilde{\alpha}_m(n_0) m 2^{\frac{1}{\lfloor m\delta \rfloor} \sum_j \tilde{M}_j},$$

a une erreur standard bornée par $\frac{0.95}{\sqrt{m}}$ lorsque m est grand.

Cette erreur standard provient d'une part de l'erreur standard de l'estimation $2^{\frac{1}{\lfloor m\delta \rfloor} \sum_j \tilde{M}_j}$, et d'autre part du calcul de la valeur de correction.

Les valeurs de corrections $\tilde{\alpha}_m$ dépendent de la valeur inconnue n, et sont ici construites à partir d'une première estimation n_0 issue de l'algorithme Loglog, à savoir

$$n_0 := \alpha_m m 2^{\frac{1}{m} \sum M_i}.$$

La fonction $\tilde{\alpha}_m(n)$ est égale à $a_n^{-m'}$ où

$$\begin{aligned} a_n \;:=\; & \frac{1}{\delta}\Gamma\left(-\frac{1}{m'}\right)\frac{1-2^{1/m'}}{\log 2} - \frac{1}{\delta}2^{\kappa_n/m'}\sum_k 2^{k/m'}\left(e^{-\frac{1}{2^{k+\kappa_n}}} - e^{-\frac{1}{2^{k-1+\kappa_n}}}\right) \\ & + \left(1 - \frac{e^{-1/2^{\kappa_n-1}}}{\delta}\right)2^{\kappa_n/m'}. \end{aligned}$$

et $\kappa_n := \left\lceil \log_2 \frac{1}{1-\delta^{m/n}} \right\rceil - \log_2(n/m) \sim \lceil \log_2(n_0/m) - \log_2 \log(1/\delta) \rceil - \log_2(n_0/m)$.

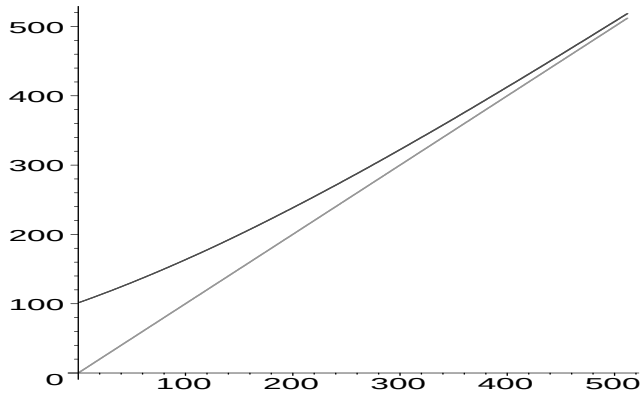

FIG. 5.3 – La moyenne de 10000 exécutions pour m=256, et $n \leq 500$. On observe la présence des non-linéarités initiales

Dans l'algorithme, cette fonction de correction est approchée par un polynôme de degré par exemple 4, $P_m(\kappa_n)$ (comme (5.2)). On observe que la fonction $\tilde{\alpha}_m(n)$ est périodique de période $\log_2 n$, ce qui justifie l'approximation polynomiale faite sur une période.

5.3 Correction des non-linéarités

La figure 5.3 présente le tracé de la moyenne de 10000 exécutions de l'algorithme Loglog, pour des petites valeurs de n (entre 0 et 512) avec $m = 256$, ainsi que le tracé de la droite $y = x$. On constate immédiatement que l'algorithme n'est pas linéaire au départ, et met un certain temps avant d'atteindre son régime asymptotique. Ce phénomène s'explique très simplement. Lorsque $n = 0$, toutes les urnes sont vides, donc tous les M_i sont égaux à 0. La formule de l'estimateur $\alpha_m m 2^{\sum Mi/m}$ donne alors $\alpha_m m$; par exemple pour $m = 256$, on trouve 101.4. Et lorsque n reste petit, ce phénomène persiste mais il devient très peu perceptible dès que $n = 2m$.

Même si l'estimateur est inexact pour des petites valeurs de n, on possède toute l'information nécessaire pour obtenir un résultat précis. En effet, lorsqu'on lance n balles (les mots) dans m urnes (selon leur préfixe), on sait estimer quelle proportion d'urnes aura reçu des balles et quelle proportion sera vide. En l'occurrence, lors de l'exécution de l'algorithme, les valeurs M_i associées aux urnes dépendent du nombre de mots reçus par chaque urne. Si l'urne i est vide, alors $M_i = 0$, et si au contraire l'urne i a reçu au moins un mot alors $M_i > 0$. On va donc utiliser la méthode de comptage de collisions présentée dans la section 4.3 pour déterminer le nombre de mots distincts rencontrés, lorsque n est faible.

On dispose de deux moyens d'estimer n : le premier est l'estimateur déjà rencontré, qui vaut $\alpha_m m 2^{\sum M_i/m}$, qui est asymptotiquement linéaire et non biaisé, mais imprécis au départ, et le second est basé sur le comptage de collision qui est $-m \log \#\{M_i = 0\}/m$, cet estimateur est très précis au départ, mais devient mauvais si n est trop grand par rapport à m. Sur une zone intermédiaire, les deux estimateurs présentent des précisions très comparables. Pour l'implantation de cette partie, j'ai fixé la limite à $5m/2$. On ne peut pas décider a priori quelle méthode utiliser, car n est inconnu, mais on peut calculer les deux estimations, et répondre suivant les deux valeurs. On note *ecc* l'estimation obtenue par comptage de collision, et *ell* l'estimation utilisée pour le Loglog classique.

- Si les deux estimations sont inférieures à $5m/2$, répondre *ecc* ;
- Si les deux estimations sont supérieures à $5m/2$, répondre *ell* ;

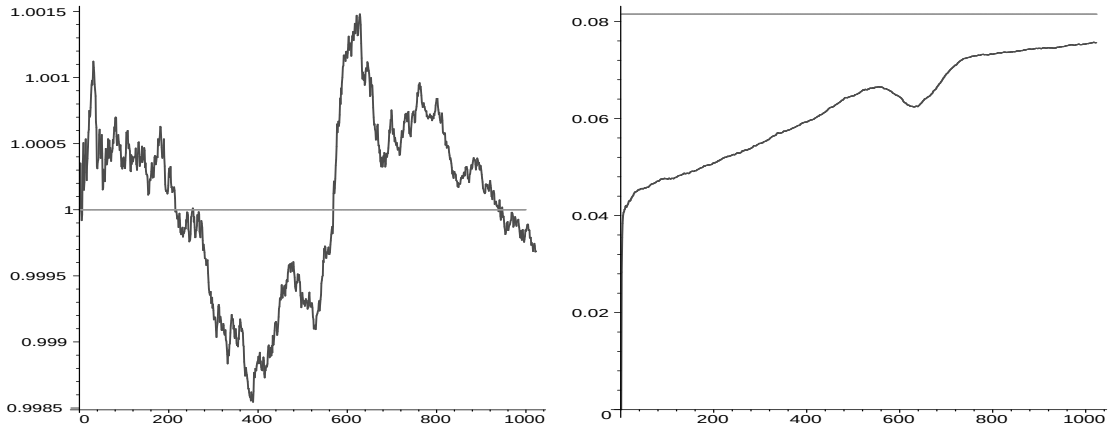

FIG. 5.4 – Tracé de la moyenne d'exécutions normalisées pour $n \leq 1024$ et $m = 256$, sur 10000 exécutions [gauche]. Tracé de l'erreur standard observée correspondante, dans les mêmes conditions [droite].

– Sinon renvoyer la moyenne arithmétique $\frac{1}{2}(ecc + ell)$.

Les résultats expérimentaux peuvent alors se lire sur la figure 5.4. Le premier tracé représente pour $m = 256$, la moyenne sur 10000 tirages aléatoires du résultat de l'algorithme divisé par n, pour n compris entre 0 et $4m$. Le deuxième tracé présente la moyenne de l'erreur standard observée dans les mêmes conditions. On voit sur le premier tracé que, dès les premières valeurs de n, le nouvel estimateur est bon, car la moyenne sur 10000 tirages ne s'éloigne pas de plus de deux pour mille de la bonne réponse. Le deuxième tracé représente la courbe expérimentale de l'erreur standard, sur lequel on voit que la précision pour les petites valeurs de n est excellente, bien meilleure même que l'erreur standard attendue asymptotiquement, qui est représentée par le trait horizontal.

5.4 Mots finis

Dans les conditions réelles d'utilisation, les algorithmes Loglog et super Loglog prennent en entrée des mots finis de longueur L, et non des mots infinis comme on l'a supposé jusqu'ici. La modification des algorithmes présentés dans les section 4.4 et 5.2 passe par la modification de la fonction ρ qui code la position du premier 1. On ajoute que lorsque w est le mot uniquement constitué de zéros, alors $\rho(w) = k + 1$, si $|w| = k$.

Ce retour à la réalité a deux conséquences qui sont étudiées dans cette section. La première est que la distribution de probabilité des registres est modifiée, et devient finie. On voit dans la première sous section que la troncature des mots est annulée par la troncature des registres de l'algorithme super Loglog, et a une influence négligeable si n ne dépasse pas certaines limites. La deuxième est que le hachage des données entraîne des collisions, en nombre non négligeable si n est grand. Ce point est traité dans la deuxième sous-section.

5.4.1 Troncature des mots

Proposition 5.1 *L'algorithme super Loglog appliqué à un ensemble de cardinal $n \leq N_{max}$ constitué de mots de taille L en bits a une erreur standard bornée par $0.95/\sqrt{m}$, à condition que les mots*

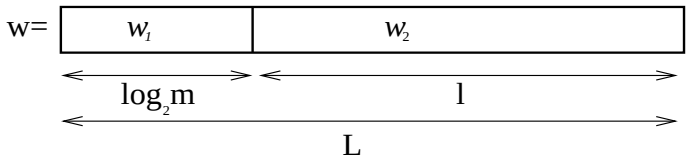

FIG. 5.5 – Notations

aient une longueur minimale vérifiant la condition

$$L \geq \log_2 m + \left\lceil \log_2 \frac{N_{max}}{m} \right\rceil + 2. \tag{5.4}$$

Notations. Les notations utilisées dans cette section sont les suivantes. L'algorithme prend en entrée des mots de longueur L, et chaque mot w est décomposé en un préfixe de longueur $\log_2 m$ noté w_1, et un suffixe de longueur $l := L - \log_2 m$, comme montré sur la figure 5.5. On note de plus $w = 0$ pour désigner un mot dont tous les bits sont à zéro. Le nombre de bits d'un mot fini w est notée $|w|$.

Pour analyser l'influence de cette troncature des mots, on suppose qu'on dispose au départ d'un ensemble E de mots binaires infinis, et on note E^f l'ensemble constitué des préfixes de longueur L des mots de E. C'est-à-dire que E^f est la partie qui est prise en compte par l'algorithme. On note M_i les maximums des premiers 1 sur l'ensemble E, ce qui est la notation utilisée dans les sections précédentes, et M_i^f, les maximums obtenus sur les mots tronqués de l'ensemble E_f. La lettre w désigne maintenant toujours un mot fini, et la lettre x toujours un mot infini.

Intuition Il y a une différence entre les valeur M_i et M_i^f, s'il existe un mot $x = x_1x_2$ de E tel que $|x_1| = \log_2 m$, $x_1 = i$ et tel que $x_2 = 0^{l+1}x_3$. Dans ce cas, la valeur de M_i est strictement supérieure à $l + 1$, alors que la valeur de M_i^f est, par définition, bornée par $l + 1$. Dans le cas de l'algorithme Loglog, le rapport entre les estimateurs $\alpha_m m 2^{1/m \sum M_i}$ et $\alpha_m m 2^{1/m \sum M_i^f}$ est alors au moins $2^{1/m}$, ce qui est non négligeable par rapport à la précision espérée.

En ce qui concerne l'algorithme super Loglog, la situation est plus favorable, car s'il y a un mot $x = x_1x_2$ tel que x_2 commence par plus de $l + 1$ zéros, alors il y aura une différence entre M_{x_1} et $M_{x_1}^f$, mais qui a peu de chance d'intervenir dans l'estimation finale car la valeur $M_{x_1}^f$ fera probablement partie des valeurs tronquées. On constate donc que l'erreur issue de la troncature des mots est en partie compensée par l'oubli des grandes valeurs M_i^f.

Lorsque n devient très grand, cette compensation n'est plus suffisante, et l'algorithme super Loglog perd en précision. On remarque en particulier que comme chaque M_i est borné par $l + 1$, l'estimation renvoyée par super Loglog est bornée par $\tilde{\alpha}_m m 2^{l+1}$. L'algorithme devient nécessairement inadéquat lorsque n est supérieur à cette valeur.

Analyse.

Choix de la longueur des mots. La valeur N_{max} correspond au nombre maximal d'éléments qu'on souhaite estimer. Comme l'erreur due à la troncature augmente avec n, on borne dans cette section l'erreur dans le cas ou il y a N_{max} éléments. Cette borne reste a fortiori valable pour le cas $n \leq N_{max}$.

Un mot induit quant au comportement des registres des conséquences différentes dans les cas fini et infini lorsque le suffixe du mot infini commence par $l + 1$ zéros. Soit $p = 1/2^{l+1}$ la probabilité

de cet événement. Le nombre de mots dont le suffixe commence par $l+1$ zéros est noté D (pour Différent). La loi de probabilité de D est

$$\Pr(D=k) = \binom{N_{max}}{k} p^k (1-p)^{n-k}.$$

Cette distribution s'approxime par une distribution poissonienne car pN_{max} est une constante et on s'intéresse au cas où k est très petit devant N_{max}. On a alors l'approximation numérique[3]

$$\Pr(D=k) \approx e^{-pN_{max}} \frac{(pN_{max})^k}{k!}.$$

La moyenne de D est égale à pN_{max}. Sachant que l'algorithme super Loglog tronque 30% des valeurs, on peut se permettre d'avoir une partie des mots qui créent une différence entre le modèle fini et infini, sans qu'il y ait au final de conséquences. On choisit, et la suite justifiera ce choix, d'imposer qu'il y ait en moyenne moins de $m/8$ mots qui posent problème. On a alors

$$pN_{max} \leq \frac{m}{8}.$$

Sachant que la longueur des mots finis est $L = \log_2 m + l$, cette inégalité se traduit en une minoration de L qui est

$$L \geq \log_2 m + \left\lceil \log_2 \frac{N_{max}}{m} \right\rceil + 2.$$

On suppose dans la suite que la taille des mots L est supérieure à $\log_2 m + \lceil \log_2 \frac{N_{max}}{m} \rceil + 2$, ce qui va permettre de borner l'erreur induite.

Majoration de l'erreur Soit un mot infini $x = x_1 x_2$ dont le suffixe x_2 commence par $l+k$ zéros, la différence entre $\rho(x_2)$ dans le cas fini et dans le cas infini est égale à k. La modification du quotient M_{x_1} par $M^f_{x_1}$ (en utilisant les notations du paragraphe précédent) due à x est inférieure à $2^{k/m}$. En moyenne, le quotient est majoré par $\frac{2^{1/m-1}}{1-2^{1/m-1}}$, qu'on majore par 1.5, en supposant que m est supérieur à 4 (le cas m strictement inférieur à 4 ferait simplement appel à une constante plus grande). Si j mots ont le même préfixe i, et tous un suffixe qui commence par au moins $l+1$ zéros, alors on majore le rapport entre M_i et M^f_i par 1.5^j, ce qui surestime la contribution réelle de ces j mots.

On note R le quotient des deux estimations, celle basée sur les mots infinis et celle basée sur les mots finis. La variable R s'exprime comme $2^{\frac{1}{m}\sum M_i - M^f_i}$, ou la somme est prise sur les 70% plus petite valeurs. On majore alors la valeur moyenne de R comme suit

$$\begin{aligned}
\mathbb{E}(R) &\leq \sum_{k \leq \lfloor 3m/10 \rfloor} e^{pN_{max}} \frac{(pN_{max})^k}{k!} + \sum_{k > \lfloor 3m/10 \rfloor} e^{pN_{max}} \frac{(pN_{max})^k}{k!} (1.5)^{k - \lfloor 3m/10 \rfloor} \\
&\leq 1 + (1.5)^{-\lfloor 3m/10 \rfloor} e^{-pN_{max}} 2 \frac{(pN1.5)^{\lfloor 3m/10 \rfloor}}{\lfloor 3m/10 \rfloor !} \\
&= 1 + a(m).
\end{aligned}$$

La fonction $a(m)$ décroît exponentiellement en m, et on vérifie que $a(64) \leq 10^{-3}$. Ceci conclut la preuve de la proposition 5.1.

[3]On utilise la notation $\approx$ pour relier des quantités dont la différence est numériquement négligeable dans les conditions du problème.

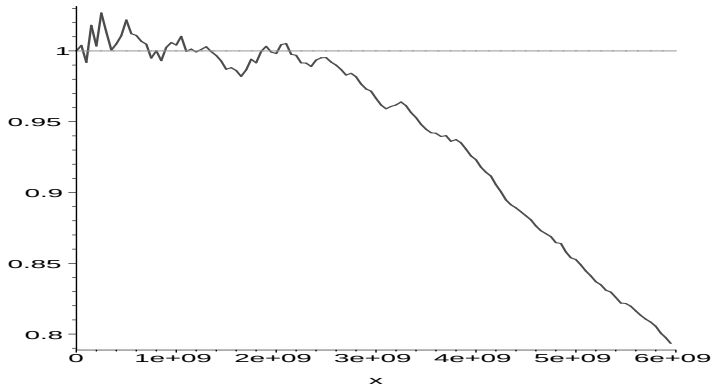

FIG. 5.6 – Une exécution de super Loglog pour $n = 6.10^9$ et $m = 2^{12}$ (espérance normalisée).

Zone de validité. La proposition 5.1 décrit la longueur des mots L qui est nécessaire pour obtenir une estimation de cardinalité. Si L est fixé, on peut inverser la formule pour en extraire la zone de validité de l'algorithme correspondante, à savoir

$$N_{max} = \frac{2^L}{4}.$$

Par exemple, des mots de 32 bits permettent d'étudier des ensembles de cardinal jusqu'à un milliard, et des mots de 64 bits des ensembles de cardinal jusqu'à quatre milliards de milliards...

La figure 5.6 montre une exécution de l'algorithme super Loglog dans un cas extrême où $n = 6.10^9$ et $m = 2^{12}$ et $L = 32$. La valeur N_{max} est alors égale à 10^9. On observe qu'au delà de la limite N_{max}, l'estimation s'éloigne complètement de la droite $y = 1$, ce qui illustre bien les limites de validité de l'algorithme.

5.4.2 Collisions

Lorsqu'on hache un ensemble de cardinal n dans l'ensemble des mots de taille L, ou de façon équivalente dans une table de hachage de taille 2^L, il va très probablement se produire des collisions. L'exemple classique est de comparer les dates d'anniversaires d'un groupe de personnes. S'il y a plus de 23 personnes, il est vraisemblable qu'au moins deux personnes aient le même anniversaire (paradoxe des anniversaires). Le nombre de ces collisions peut être estimé de manière très précise si n n'est pas trop grand par rapport à 2^L, comme on l'a montré dans la section 5.4. Or la proposition 5.1 impose la limite $n \leq \frac{2^L}{4}$ pour que l'algorithme super Loglog garde la précision souhaitée. Le biais crée par ces collisions est donc facilement rectifié.

Notons n le nombre d'éléments distincts de l'ensemble initial, et $\tilde{n}$ le nombre d'éléments distincts après hachage (qui est la valeur estimée par l'algorithme). Le fait qu'il puisse y avoir des collisions implique $\tilde{n} \leq n$. La section 4.3 a montré que $n_1 = -2^L \log(1 - \tilde{n}2^{-L})$ est un approximant acceptable de n. Pour corriger l'effet des collisions, il suffit donc de modifier l'algorithme en conséquence.

Erreur. On veut ici estimer l'erreur apportée par le fait qu'il y ait des collisions dans la fonction de hachage. Cette erreur a deux origines. Tout d'abord le hachage lui-même, et ensuite la correction effectuée par l'algorithme.

L'erreur due au hachage est $\frac{1}{\sqrt{2^L}}\sqrt{e^\rho - 1}$, où ρ désigne le remplissage de la table, qui est borné par $1/4$. Cette erreur est majorée par $\frac{1}{\sqrt{m}}\frac{0.6}{\sqrt{2^l}}$, où l est la longueur du suffixe pris en compte ($L =$

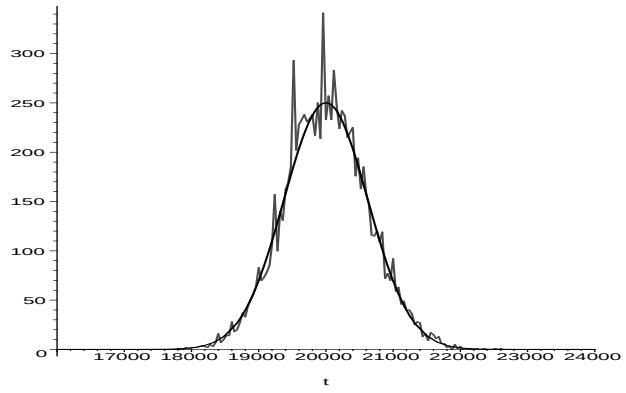

FIG. 5.7 – Distribution de l'estimateur

$\log_2 m+l$). Si l est raisonnablement grand, l'erreur crée par le hachage est négligeable devant l'erreur de l'algorithme ($\frac{1.3}{\sqrt{m}}$ pour Loglog).

L'erreur due à l'estimation est donnée dans la section 4.3, et est égale à $\frac{e^\rho-1}{n}$, ce qu'on majore ici par $\frac{1}{\sqrt{m}}\frac{1.2}{\sqrt{2^l}}$.

L'erreur conjuguée de ces deux phénomènes est finalement majorée par $\frac{1}{\sqrt{m}}\frac{2}{\sqrt{2^l}}$. Pour le cas particulier $L = 32$ et $m = 1024$, l'erreur due au hachage est inférieure à 3.10^{-5}, alors que l'erreur due à l'algorithme est environ 4%.

5.5 Distribution de l'estimateur

On revient ici à l'algorithme Loglog, dont on souhaite estimer les risques d'erreur.

Proposition 5.2 *La distribution de la variable aléatoire e_n définie par $e_n = \alpha_m 2^{\frac{1}{m}\sum M_i}$ est asymptotiquement gaussienne lorsque m tend vers l'infini :*

$$\frac{e_n - n}{n\beta_m/\sqrt{m}} \to \mathcal{N}(0,1).$$

Preuve. (Esquisse) La preuve de cette proposition se base sur le théorème central limite : lorsqu'on dispose de m variables aléatoires indépendantes et identiquement distribuées de moyenne μ et de variance σ^2, alors la variable aléatoire Z_m définie par $Z_m := \frac{\sum X_i - m\mu}{\sigma\sqrt{m}}$ converge vers la loi normale $\mathcal{N}(0,1)$, lorsque m tend vers l'infini.

La distribution de tous les registres M_i est la même, à des variations stochastiques près, liées aux cardinalités des groupes et qui sont négligeables. On note μ l'espérance des M_i, et σ^2 la variance correspondante. D'après le théorème central limite énoncé précédemment, on obtient que $\frac{1}{m}\sum M_i$, après normalisation tend vers la loi normale, c'est-à-dire

$$\frac{1}{m}\sum M_i = \mu + \frac{\sigma}{\sqrt{m}}G_m,$$

où G_m est une variable aléatoire qui converge en loi vers G telle que $G \stackrel{d}{=} \mathcal{N}(0,1)$ ($\stackrel{d}{=}$ représente l'égalité en distribution).

Soit ensuite la variable aléatoire Y définie par $Y = 2^{\frac{1}{m}\sum M_i}$. On peut alors écrire

$$Y \stackrel{d}{=} 2^{\mu+\frac{\sigma}{\sqrt{m}}G_m}.$$

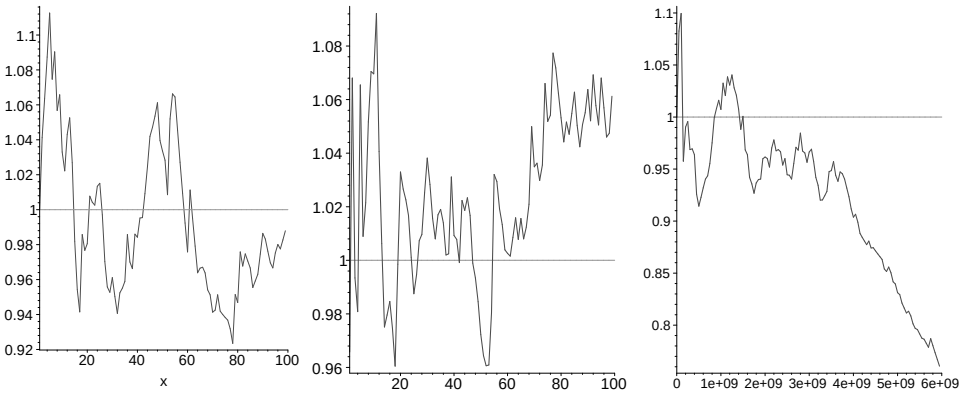

FIG. 5.8 – Une exécution de super Loglog pour $n = 10^3$, $n = 10^6$, $n = 6.10^9$ et $m = 2^8$ (espérance normalisée).

L'écart-type σ est une constante, et m tend vers l'infini, ce qui permet l'approximation suivante

$$Y \stackrel{d}{=} 2^{\mu} \left(1 + \frac{\sigma}{\sqrt{m}} G_m + o\left(\frac{1}{m}\right)\right).$$

Asymptotiquement, la variable aléatoire Y est donc gaussienne, et il en est de même pour l'estimateur e_n car $e_n = \alpha_m Y$.

On a montré dans le chapitre 4 que l'espérance de l'estimateur e_n est asymptotiquement n (à des fluctuations extrêmement petites près), et que sa variance est $\frac{n^2 \beta_m^2}{m}$, également à des fluctuations près. Cela conclut la preuve de la proposition. ■

Remarque. Plus précisément, l'exponentielle d'une loi normale est une loi log-normale. Ici, la loi limite est normale car l'écart-type est asymptotiquement négligeable devant la moyenne.

La distribution de l'estimateur renvoyé par l'algorithme super Loglog est également gaussienne, car les arguments de la preuve précédente demeurent valables. Les deux algorithmes ont la même moyenne n, par contre la variance de l'estimateur de l'algorithme super Loglog est inférieure à celle de l'estimateur de l'algorithme Loglog.

Le fait de savoir que la distribution limite est gaussienne permet de quantifier l'erreur commise selon l'écart-type $\sigma = \sqrt{\frac{n^2 \beta_m^2}{m}}$. Ainsi la probabilité d'être à une distance de la moyenne inférieure à σ, 2σ, 3σ est respectivement de 65%, 95% et 99%.

La figure 5.7 présente l'histogramme de la distribution observée sur 1000 estimations, correspondant à 1000 exécutions de l'algorithme super Loglog, pour les valeurs $m = 10$ et $n = 20000$, ainsi que la courbe gaussienne attendue.

5.6 Résultats expérimentaux

5.6.1 Super Loglog - données aléatoires

Dans cette sous-section, l'algorithme est appliqué à des données aléatoires de 32 bits engendrées par un générateur pseudo-aléatoire de qualité. La figure 5.8 montre trois exécutions de l'algorithme, pour la valeur $m = 2^8$, et des valeurs de n petites ($n \leq 1000$) moyennes ($n \leq 10^6$) et très grandes ($n \leq 6.10^9$). Pour le choix $m = 2^8$, l'erreur standard prédite est 6.2%. On constate sur les deux

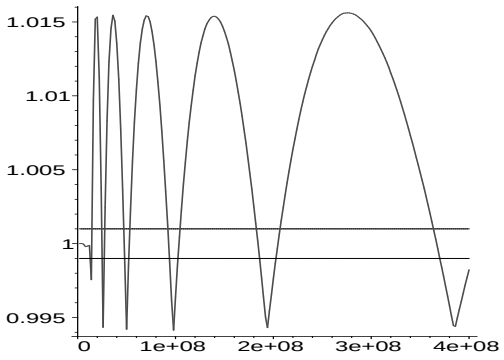

FIG. 5.9 – Tracé de $\alpha m 2^{\frac{1}{\delta m}\sum \tilde{M}_i}$ pour $n = 10^8$ et $m = 2^{20}$ sur des données aléatoires, moyenné sur 100 exécutions de l'algorithme.

premiers graphes de la figure 5.8 que l'estimation est la plupart du temps comprise entre $1 - 0.062$ et $1 + 0.062$, même si, comme il est attendu, il lui arrive de sortir de cette zone. Le troisième graphe par contre correspond à une exécution de l'algorithme en partie hors de sa zone de validité, on constate donc que l'estimation diminue et perd en qualité lorsque n dépasse la limite N_{max} qui est ici égale à 10^9.

5.6.2 Importance de la constante de correction $\tilde{\alpha}_m$

La figure 5.9 montre la tracé de $\alpha m 2^{\frac{1}{\delta m}\sum \tilde{M}_i}$ où α est une constante fixée. On voit très bien sur cette figure que pour les grands fichiers, il y a des oscillations dont la période augmente avec n de façon logarithmique.

5.6.3 Erreur standard

Les courbes de la figure 5.10 montrent le tracé de l'erreur standard observée pour $n = 10^6$, m compris entre 2^4 et 2^{20} et 1000 exécutions, en comparaison avec l'erreur standard théorique. La courbe de gauche correspond au cas de l'algorithme Loglog, et celle de droite au cas de l'algorithme super Loglog. Dans les deux cas la courbe expérimentale est légèrement supérieure à la courbe théorique pour les petites valeurs de m, mais on reconnaît aisément le comportement en $1.3/\sqrt{m}$ pour Loglog et $0.95/\sqrt{m}$ pour super Loglog.

Pour le cas de l'algorithme super Loglog, on observe que l'écart entre la courbe expérimentale et la courbe théorique est de l'ordre de grandeur de $\frac{1}{m}$. Cet écart est négligeable asymptotiquement, mais explique le léger écart pour des valeurs de m petites.

5.6.4 Super Loglog - données réelles

Textes en français ou anglais

Les textes en anglais ou en français sont typiquement des données où les mots ne sont pas distribués de façon aléatoire. Il y a des mots très présents, et d'autres très peu. Par exemple dans Tartuffe de Molière, le mot "vous" apparaît 580 fois, alors qu'on ne lit qu'une fois le mot "vermeille". De plus l'apparition d'un mot dépend bien sûr des mots précédents. Dans Hamlet le mot "be" apparaît 32 fois après "to", et 217 fois au total. L'ordre de grandeur de la quantité de mots différents dans un livre en français ou en anglais est 10 000, en comptant les formes (accordées, conjuguées)

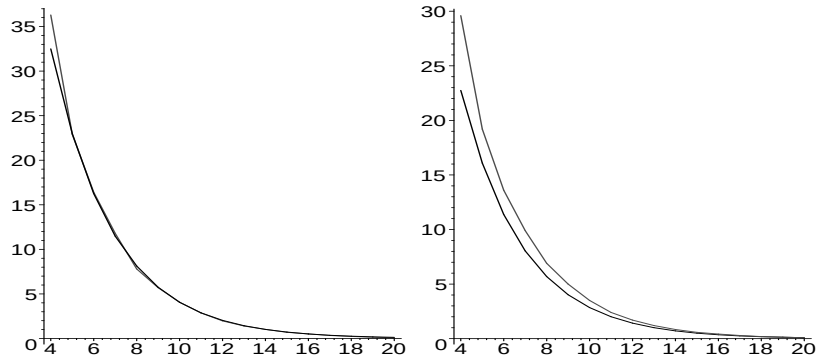

FIG. 5.10 – Tracé de l'erreur standard expérimentale et théorique, pour l'algorithme Loglog [gauche] et super Loglog [droite].

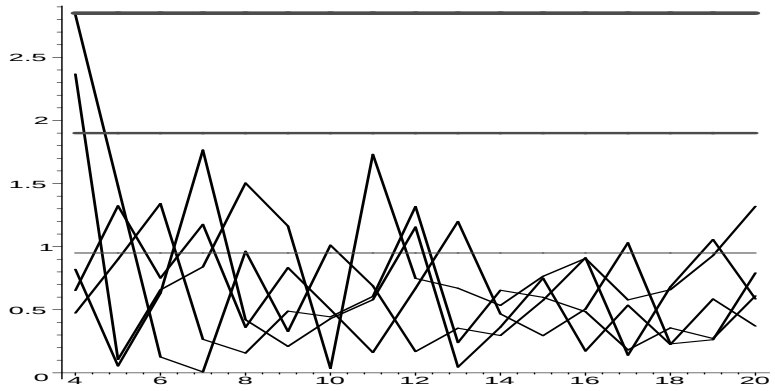

FIG. 5.11 – Tracé de l'erreur multipliée par $\sqrt{m}$ pour 5 textes différents. Les traits horizontaux sont positionnés respectivement à un, deux et trois écarts-types.

des mots, et varie significativement selon l'auteur et la longueur du livre. Par exemple "Les trois mousquetaires" contient 14822 mots différents, "Le tour du monde en 80 jours" 9441 et "Hamlet" qui est le plus court 5044.

La figure 5.11 présente le tracé de 5 courbes correspondant à 5 textes différents : la bible, le tour du monde en 80 jours, Hamlet, Tartuffe et les trois mousquetaires. Pour chaque valeur de $\log_2 m$ comprise entre 4 et 20, on trace la valeur $|Est - Real|\sqrt{m}$, ou Est est l' estimation et $Real$ la valeur réelle. On fait figurer de plus trois traits horizontaux situés à 0.95, 1.90 et 2.85, qui correspondent après normalisation à une, deux et trois erreurs standard. On s'attend donc, en suivant la remarque de la section 5.5 à ce que 65% des valeurs soient situées en dessous du premier trait horizontal, 95% en dessous du deuxième, et 99% en dessous du troisième. Cela est confirmé visuellement par ce qu'on observe sur cette figure.

Regveda et Mahabharata

Le regveda est un texte indien écrit en sanscrit qui date du deuxième millénaire avant JC. Je ne résiste pas à l'envie d'en citer quelques vers, extraits du récit de la naissance divine de la déesse Indra.

4.018.01a ayám pánthā ánuvittaḥ purāṇó yáto devā́ udájāyanta víśve

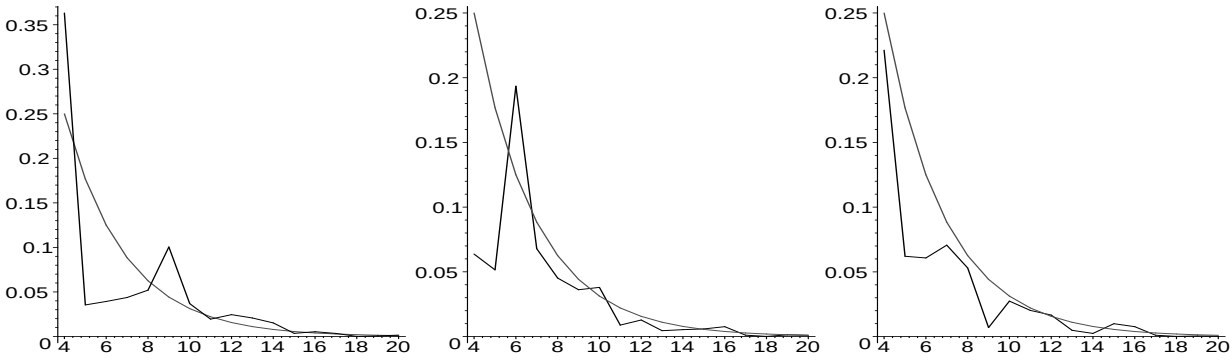

FIG. 5.12 – Erreur relative de l'estimateur pour le texte du Mahabharata, comparée à l'erreur standard théorique ($m = 2^4, \ldots, 2^{20}$), pour trois fonctions de hachage différentes.

4.018.01c átaś cid ā́ janiṣīṣña právṛddho mā́ mātáram amuyā́ páttave kaḥ
4.018.02a nā́hám áto nír ayā durgáhaitát tiraścátā pārśvā́n nír gamāṇi
4.018.02c bahū́ni me ákṛtā kártvāni yúdhyai tvena sáṃ tvena pṛchai
Soit en anglais :

This is the ancient and accepted pathway by which all the gods have come into existence.
Hereby one could be born, through waxen mighty.
Not this way I go forth, hard is the passage ; forth from this side, obliquely, I will issue.
Much that is yet undone I will accomplish. One must I combat and the other question.

Ce texte a la particularité de contenir beaucoup de formes de mots distincts, car outre la richesse du style, la langue sanscrite possède une très riche morphologie et fait largement appel aux mots composés. Cela fournit ainsi un texte avec un vocabulaire abondant.

Le texte contient 45597 mots distincts pour environ 182 000 mots au total, et selon les valeurs de m, les estimations fournies par l'algorithme sont

$\log_2 m$	4	5	6	7	8	9	10	11
estimation	28827	35426	42547	41645	45444	43781	46735	46518
Précision en %	37	22	6.7	8.7	0.4	4.0	2.5	2.0

$\log_2 m$	12	13	14	16	18	20
estimation	45591	45133	45543	45416	45544	45586
Précision en %	0.02	1.0	0.12	0.4	0.12	0.03

Le Mahabharata (prononcer Ma-haa-BHAAR-a-ta) est un texte religieux indien. Il a existé sous de nombreuses formes, la plus fondamentale étant un texte en ancien Sanskrit, qui est peut être le plus grand livre du monde. Il contient environ un million de mots, dont 177601 différents. La figure 5.12 représente des courbes de l'erreur relative de l'estimation renvoyée par l'algorithme super Loglog, selon la valeur de m, pour trois fonctions de hachage différentes.

5.6.5 Traces de serveur web et π

L'algorithme super Loglog peut être appliqué à des données de taille plus importantes, telles des logs de pages web de taille $400Mb$ contenant 1.8 millions de "mots" distincts, ou encore aux 200 millions premiers digits de π, groupés 10 par 10 pour former 20 millions de mots dont sont 19979352 distincts.

Le premier tableau montre l'erreur relative (en %) des exécutions de l'algorithme pour des valeurs de m allant de 2^5 à 2^{18}, et sur un fichier de logs de pages webs contenant 1.8 millions de mots distincts. (E.S. désigne l'erreur standard attendue.)

$\log_2 m$	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Rés	12	8.9	1.2	0.23	3.5	2.6	1.9	2.5	1	1.2	1.3	0.65	0.5	0.32
E. S.	16	11	8	5.7	4.0	2.8	2	1.4	1	0.7	0.5	0.36	0.25	0.18

Le deuxième tableau montre le résultat de trois exécutions de l'algorithme sur 20 millions de mots formés par les 200 millions premiers digits de π, chacune utilisant une fonction de hachage différente, et la quatrième ligne montre l'erreur moyenne commise sur ces trois exécutions. Finalement la dernière ligne rappelle la valeur de l'erreur standard (E.S.) attendue. Toutes les valeurs sont exprimées en pourcentage.

$\log_2 m$	8	9	10	11	12	13	14	15	16	17	18	19	20
Exéc 1	0.7	1.5	2.8	0.27	1.3	0.64	0.4	**0.7**	0.22	**0.39**	**0.19**	0.13	0.06
Exéc 2	**7.2**	0.8	0.22	0.52	0.07	0.75	0.67	0.2	0.2	0.02	0.09	0.06	**0.12**
Exéc 3	4.1	2.6	0.8	1.6	**1.7**	1.0	**1.1**	0.5	0.13	0.06	0.18	0.13	**0.19**
Moy	4	1.7	1.3	0.8	1	0.8	**0.8**	0.5	0.28	0.17	0.16	0.11	**0.12**
E.S.	5.7	4.0	2.8	2	1.4	1	0.7	0.5	0.36	0.25	0.18	0.13	0.09

Les valeurs qui sont au delà d'une erreur standard de la valeur attendue sont présentées en gras. On constate que 20% des valeurs sont à plus d'un erreur-type d'écart, ce qui est cohérent vis à vis de l'approximation gaussienne de la distribution des valeurs.

5.6.6 Super Loglog - Zone de troncature

L'algorithme super Loglog présenté dans ce chapitre est basé sur un choix de troncature des registres à 70%. Ce choix est illustré par des simulations dans cette partie.

L'évolution de la précision de l'algorithme selon la valeur de troncature δ choisie repose sur deux phénomènes qui ont des effets opposés, d'une part la variance de la quantité $2^{\frac{1}{m}\sum \tilde{M}_i}$, et d'autre part l'amplitude de la valeur de correction $\tilde{\alpha}_m$. On prend δ compris entre 0.5 et 1. De plus cette zone encadre bien la valeur que nous avons choisie, qui est $\delta = 0.7$.

Erreur standard. La variance de la quantité $2^{\frac{1}{m}\sum \tilde{M}_i}$ décroît avec δ. Lorsque $\delta = 1$, l'erreur standard multipliée par $\sqrt{m}$ est égale à 1.3, et diminue jusqu'à être proche de 0.79 pour $\delta = 0.5$. Ces valeurs sont présentées dans la section 5.8.3, voir la figure 5.22, ou le tableau de valeurs (5.8). Selon ce seul critère, on a intérêt à choisir δ relativement petit, par exemple $\delta = 0.5$.

Amplitude de la valeur de correction. Lorsque δ diminue l'amplitude de l'espérance de la quantité $2^{\frac{1}{m}\sum \tilde{M}_i}$ (et par conséquent l'amplitude de la valeur de correction) augmente de façon sensible. Ce phénomène est illustré par la figure 5.8, sur laquelle on peut voir que entre $\delta = 0.9$ et $\delta = 0.5$, l'amplitude a environ doublé. Cela pose un problème, car le calcul de la valeur de correction est basé sur une valeur approchée de n. L'erreur commise sur cette valeur approchée est plus amplifiée lorsque δ est petit. Ce phénomène pousse donc vers des valeurs de δ grandes.

Bilan La conjonction de ces deux phénomènes conduit à choisir comme valeur de troncature $\delta = 70\%$. Ce choix n'est bien sur pas le seul possible, et opter pour une valeur de δ comprise entre 0.5 et 1 permet d'améliorer la précision de l'algorithme Loglog.

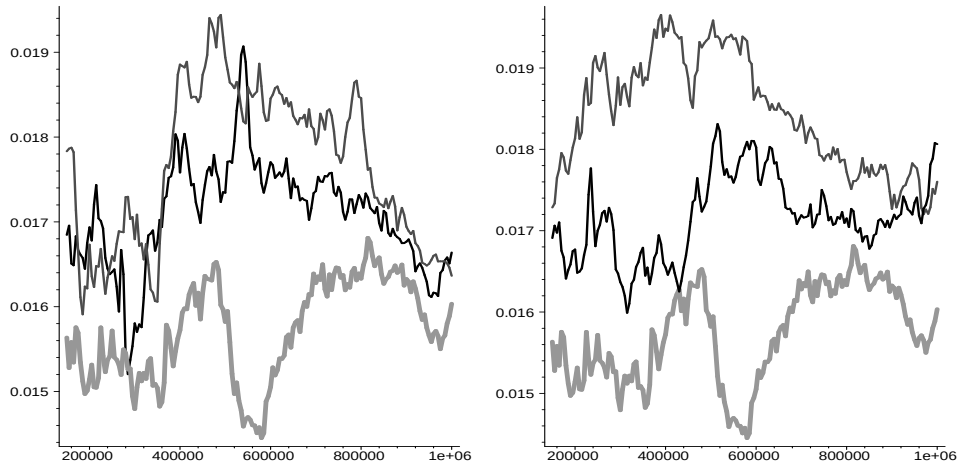

FIG. 5.13 – Comparaison des différents choix de valeur de troncature δ. A gauche, de bas en haut $\delta = 0.7$, $\delta = 0.6$, $\delta = 0.5$, et à droite bas en haut $\delta = 0.7$, $\delta = 0.8$, $\delta = 0.9$.

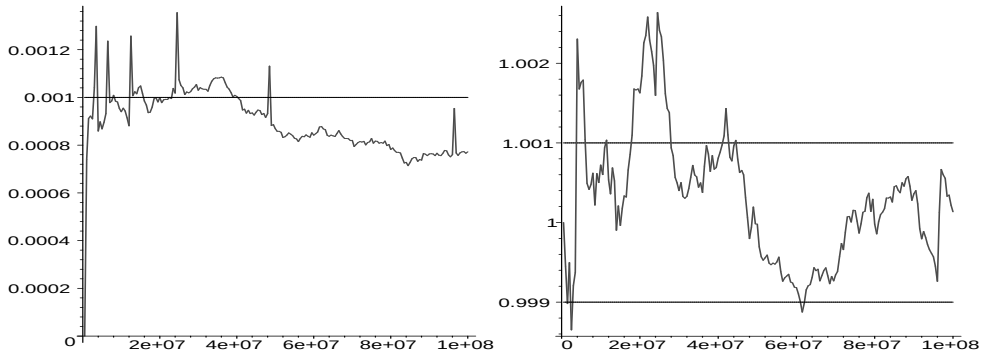

FIG. 5.14 – Tracé de l'erreur standard et d'une exécution pour des données aléatoires avec les paramètres $n = 10^8$ et $m = 2^{20}$. Les lignes horizontales indiquent le point de repère de l'erreur standard attendue.

Illustration La figure 5.13 montre les tracés de courbes représentant l'erreur standard commise pour différentes valeurs de δ. Toutes ces simulations sont obtenues pour $m = 2^{12}$, $n \leq 10^6$, et pour 200 exécutions. Le tracé de gauche montre, de bas en haut, les courbes correspondant à $\delta = 0.7$, $\delta = 0.6$, $\delta = 0.5$, et le tracé de droite montre, de bas en haut, les courbes correspondant à $\delta = 0.7$, $\delta = 0.8$, $\delta = 0.9$. On observe sur cet exemple que la valeur de troncature optimale est $\delta = 0.7$.

5.6.7 Super Loglog - précision à 0.1%

La précision de l'algorithme super Loglog peut en théorie être rendue aussi petite que désiré, en ajustant le nombre de registres. Par exemple pour obtenir un algorithme précis à 0.1%, on choisit m supérieur à la solution de l'équation $10^{-3} = \frac{0.95}{\sqrt{m}}$, soit $m \geq 9.2.10^5$. Comme on veut en plus que m soit une puissance de 2, on choisit $m = 2^{20}$. La figure 5.14 représente les traces de l'erreur standard pour 100 exécutions de l'algorithme, avec $m = 2^{20}$ et $n = 10^8$, et la trace d'une exécution normalisée. On constate donc que l'erreur standard est bien en moyenne inférieure à 0.1%.

Lorsque $m = 2^{20}$, sachant que la distribution des valeurs est approximée par une gaussienne,

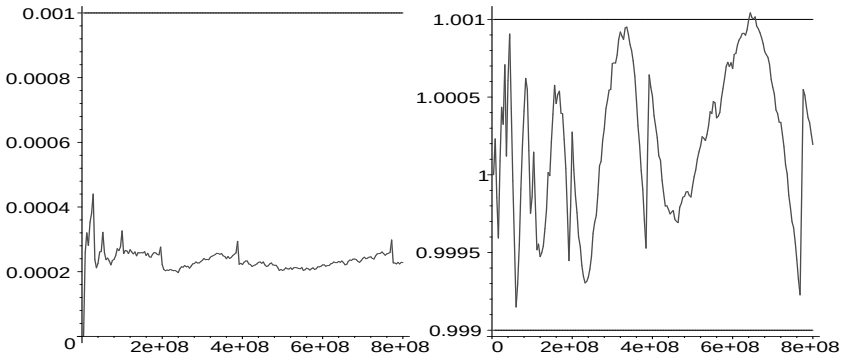

FIG. 5.15 – Tracé de l'erreur standard et d'une exécution pour des données aléatoires avec les paramètres $n = 10^8$ et $m = 2^{23}$. Les lignes horizontales indiquent le point de repère de l'erreur standard attendue.

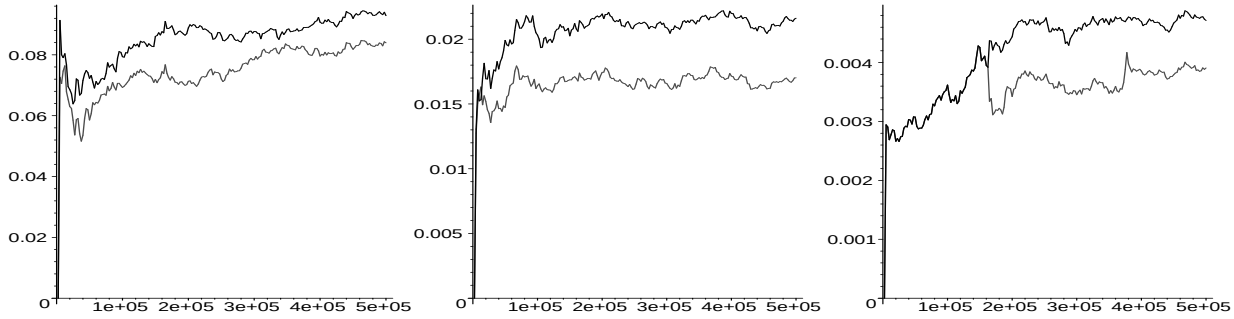

FIG. 5.16 – Comparaison de l'erreur standard expérimentale des algorithmes Loglog et super Loglog sur 100 exécutions, pour les valeurs $n = 500000$, et $m = 2^8$, 2^{12} et 2^{16}.

il y a environ 65% des exécutions qui auront une erreur inférieure à 0.1%. Pour augmenter cette proportion, on choisit une valeur de m encore plus grande. Par exemple si on prend $m = 2^{23}$, l'erreur standard attendue est 3.10^{-4}. L'approximation gaussienne dit alors que dans 99% des cas l'erreur du résultat est inférieure à 0.1%. On obtient alors un algorithme qui garantit que sauf pour un petit nombre de cas, le résultat renvoyé est à moins de 0.1% de la réalité. La figure 5.15 présente les mêmes graphes que la figure 5.14, mais pour les valeurs $n = 8.10^8$ et $m = 23$. On observe bien que l'estimation est presque toujours située à moins de 0.1% de distance du résultat attendu.

5.6.8 Loglog versus super Loglog

La figure 5.16 montre des exécutions des algorithmes Loglog et super Loglog sur les mêmes données. La figure montre le tracé de l'erreur standard dans les deux cas, pour les données suivantes : 500 000 éléments, 100 exécutions, et m égal à 2^8, 2^{12} et 2^{16}. Pour les trois tracés, la courbe la plus haute correspond à l'algorithme Loglog, et la plus basse à l'algorithme super Loglog. On observe donc expérimentalement que la version super Loglog est plus précise que la version Loglog.

On constate que sur le troisième tracé ($m = 2^{16}$), les deux courbes coïncident au départ. Cela provient du fait que pour des petites valeurs de n ($n < 5m/2$), les deux algorithmes utilisent la même méthode de correction de non linéarités, et renvoient donc exactement le même résultat.

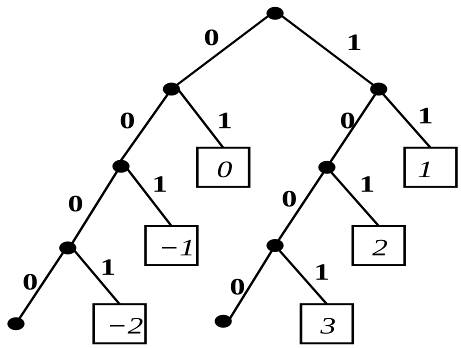

FIG. 5.17 – L'arbre de codage des données

5.7 Compression des registres

5.7.1 Algorithme de compression

Les algorithmes Loglog et super Loglog sont basés sur la mise à jour de m registres, puis sur l'évaluation d'une estimation à partir des valeurs de ces registres. Ces registres contiennent a priori des valeurs comprises entre 0 et $\log N_{max}/m$, qui sont donc codées sur environ $\log\log N_{max}$ bits, d'où le nom des deux algorithmes. Par exemple, si on fixe $N_{max} = 2^{60}$, ce qui est largement suffisant pour la plupart des applications, les valeurs des registres sont codées sur 6 bits, comme on l'a montré dans la section 5.4.1.

Cependant, si on observe la figure 4.4 qui montre les valeurs des registres après exécution de l'algorithme sur l'oeuvre complète de Shakespeare ($n = 28239$), en utilisant $m = 256$ registres, on constate que seulement sept valeurs différentes sont utilisées, dont une une seule fois. Cela signifie que les registres peuvent se compresser efficacement, sans perdre d'information, et sans perturber l'exécution de l'algorithme, ce qui permet de gagner beaucoup d'espace mémoire.

La distribution des données suit la loi de probabilité (pour $k < b_n$)

$$\Pr(M = k) = \left(1 - \frac{1}{2^k}\right)^{\nu} - \left(1 - \frac{1}{2^{k-1}}\right)^{\nu}$$

où ν est le nombre d'éléments associés à l'urne, qui vaut environ n/m. La valeur $\Pr(M = b_n)$ est le complément des probabilités précédentes. La figure 5.1 présente un exemple de cette distribution. La plupart des valeurs sont concentrées autour de la valeur $\log_2 n/m$, et en pratique seules les quelques valeurs autour de cette limite vont intervenir. Les registres sont par définition bornés, car ils représentent la position du premier "1" dans un mot de taille finie. Cela est expliqué en détail dans la section 5.4.1. On note R_{max} la valeur maximale des registres.

Le codage choisi pour coder les valeurs des registres est un codage préfixe qui va coder l'écart entre la valeur et la valeur "milieu" qui est $\lfloor \log_2 n + 1 \rfloor$, comme représenté sur la figure 5.17. Le codage 00^k1 correspond à la valeur $\lfloor \log_2 n + 1 \rfloor - k$, et ce pour $k \geq 0$, et le codage 10^k1 correspond à $\lfloor \log_2 n + 1 \rfloor + k + 1$. Toutes les valeurs possibles sont ainsi codées, et le fait d'utiliser un codage préfixe permet de concaténer toutes les valeurs $M_0, \ldots M_{m-1}$ en une chaîne de bits, sans avoir à mettre de séparateur. L'espace mémoire occupé par l'algorithme est donc la somme de l'espace mémoire occupé par tous les registres.

L'espace mémoire moyen occupé par les m registres, noté S est égal à

$$S = m \sum_{k=0}^{\lfloor \log_2 n+1 \rfloor} p_k(\lfloor \log_2 n + 1 \rfloor - k + 2) + m \sum_{k=\lfloor \log_2 n+1 \rfloor + 1}^{R_{max}} p_k(k - \lfloor \log_2 n + 1 \rfloor + 1).$$

Cette somme code le fait que les valeurs k inférieures à la limite $\lfloor \log_2 n + 1 \rfloor$ sont codées sur $k - \lfloor \log_2 n + 1 \rfloor + 2$ bits et apparaissent avec une probabilité p_k, alors que les valeurs k strictement supérieures à $\lfloor \log_2 n + 1 \rfloor$ sont codées sur $\lfloor \log_2 n + 1 \rfloor - k + 1$ et apparaissent également avec une probabilité p_k. Si on supprime la troncature R_{max} dans la deuxième somme de l'équation précédente, alors on obtient un majorant de S.

Remarque. Lors de l'exécution de l'algorithme, la valeur exacte de n, le nombre d'éléments distincts est inconnu. Cependant le fait de connaître une estimation suffit pour mettre en place en pratique cette méthode de compression des registres.

Le coût S dépend de n et de m, et est majoré par l'expression suivante

$$S \leq m \sum_{k=0}^{\lfloor \log_2 n+1 \rfloor} p_k(\lfloor \log_2 n + 1 \rfloor - k + 2) + m \sum_{k=\lfloor \log_2 n+1 \rfloor + 1}^{\infty} p_k(k - \lfloor \log_2 n + 1 \rfloor + 1).$$

La somme qui majore S est périodique en $\log_2 n$, ce qui fait qu'on peut se contenter de l'étudier numériquement sur un intervalle de type $[x, 2x]$. On obtient alors

$$S \leq 3.01m.$$

On a obtenu une majoration de l'espace moyen occupé par l'algorithme, S. Il reste encore à trouver une valeur S_{max}, telle qu'avec une très grande probabilité, l'espace mémoire occupé par l'algorithme soit inférieur à S_{max}. L'écart entre le nombre attendu mp_k de registres égaux à k, et les valeurs constatées est de l'ordre de $\sqrt{m}$. L'écart entre S et S_{max} est par définition du même ordre de grandeur, et donc lorsque m est suffisamment grand, choisir S_{max} égal à αm avec α une constante suffisamment supérieure à 3.01 convient. On choisit

$$S_{max} = 3.2m.$$

Ce choix est justifié expérimentalement dans la section suivante.

On constate que cette compression des registres n'affecte pas l'algorithme, même s'il y a un surcoût lorsque la partie entière de $\log_2 n/m$ change car il faut alors modifier toutes les valeurs. Ce phénomène reste cependant extrêmement rare, et peut être négligé. En dehors de ce cas de figure, la mise à jour des registres est faite sans difficulté supplémentaire.

La précision de l'algorithme est alors amélioré de 20% par rapport à un codage des registres sur 5 bits, et de 27% par rapport à un codage sur 6 bits. Ce qui revient à dire que si l'on impose $N_{max} = 10^{18}$, en utilisant la même quantité de mémoire que dans l'algorithme super Loglog sans compression, on obtient une précision qui est

$$\text{Erreur standard} = \frac{0.7}{\sqrt{m}}.$$

5.7.2 Expériences

La figure 5.19 montre le taux de compression moyen de chaque registre pour différentes valeurs de m, sur 100 exécutions et pour $n = 100000$.

La figure 5.19 montre le même paramètre, mais sur une seule exécution

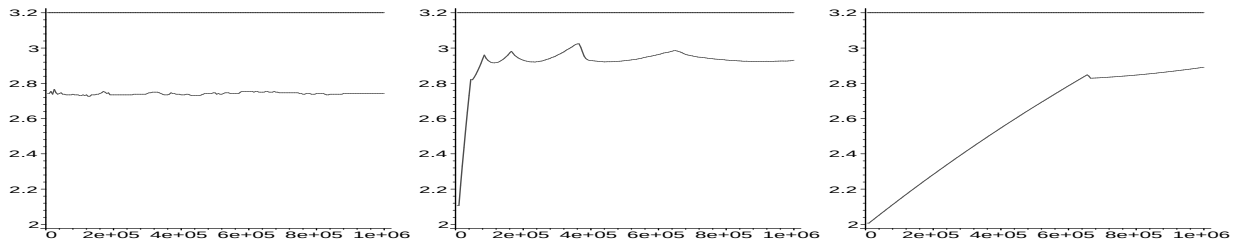

FIG. 5.18 – Mémoire moyenne occupée par un registre selon le nombre d'éléments. Expérience faite sur 100 exécutions, $n = 100000$, et (a) $m = 2^8$, (b)$m = 2^{12}$ et (c)$m = 2^{16}$

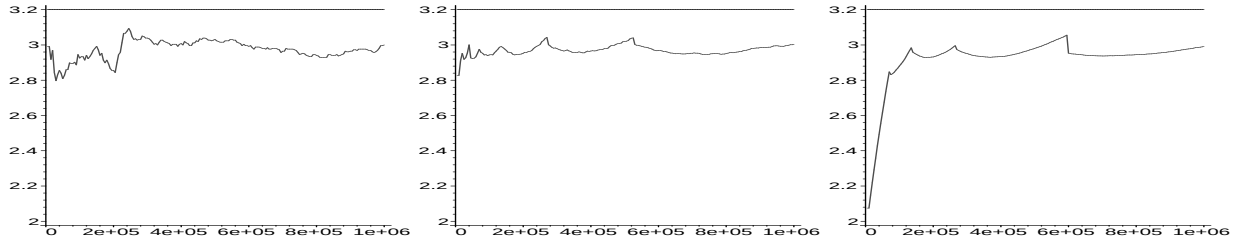

FIG. 5.19 – Même chose que pour la figure 5.18, mais sur une seule exécution.

5.8 Analyse de super Loglog

L'analyse de l'algorithme super Loglog utilise les mêmes outils que celle de l'algorithme Loglog, mais de manière plus complexe, car la distribution des registres est désormais tronquée. On se base donc dans cette section sur quelques hypothèses simplificatrices, comme par exemple l'équirépartition des éléments dans les urnes, propriété qui est valide au premier ordre.

La mise en équation de l'espérance et de la variance est effectuée en 5.8.1. Les sous-sections 5.8.2 et 5.8.3 étudient le comportement asymptotique de ces deux quantités. La sous-section 5.8.4 détaille la valeur de l'erreur standard. La sous-section suivante 5.5 est consacrée à l'étude de la distribution de l'estimation calculée par l'algorithme super Loglog.

Dans cette section, on reprend les notations utilisées dans les sections précédentes, à savoir le nombre d'éléments distincts à évaluer est noté n, et le nombre de registres utilisé m. Les registres sont notés M_i et les δ plus petits registres sont notés $\tilde{M}_i$.

On rappelle que les énoncés nommés *Résultat* sont basés sur quelques hypothèses simplificatrices (toujours extrêmement réalistes).

5.8.1 Intuition - Mise en équation

Pour l'algorithme Loglog, lorsqu'on évalue n éléments grâce à m registres, la probabilité qu'un registre ait une valeur inférieure à k est

$$\Pr(M \leq k) = \left(1 - \frac{1}{2^k}\right)^{\nu}, \tag{5.5}$$

où ν est le nombre d'éléments dirigés vers l'urne, voir l'équation (4.1).

Lorsqu'on décide de tronquer les valeurs les plus grandes, la loi de probabilité change, les valeurs les plus petites deviennent plus probables, et la distribution devient plus concentrée. On note δ la fraction de valeurs que l'on souhaite conserver, et on considère la distribution de probabilité d'une

variable notée L_1 conditionnée à ne conserver que la fraction δ la plus à gauche de la distribution initiale. Une telle distribution tronquée est représentée sur la figure 5.2.

La troncature des valeurs les plus élevées revient à faire m tirages selon la distribution initiale, et à ne garder que les δm plus petits. Lors de ces tirages , il y a $m\delta$ valeurs qui tombent dans la zone correspondant à la distribution L_1, à un facteur $1 + O(1/\sqrt{m})$ près. On considère donc pour super Loglog le modèle selon lequel on lance δm valeurs selon la distribution L_1 (qui est valide à un facteur $1 + O(1/\sqrt{m})$ près). C'est maintenant ce second modèle, plus simple, qu'on analyse. Pour bien distinguer le nombre d'urnes noté m du nombre d'urnes prises en compte par super Loglog, on note $m' = \delta m$.

Mise en équation

Hypothèse H1 : il y a m' urnes qui reçoivent chacune n/m éléments, et qui suivent la loi de probabilité L_1.

Cette hypothèse est valide au premier ordre asymptotique, car le nombre d'éléments dans chaque urne est $\frac{n}{m}\left(1 + O(n^{-1/2})\right)$. Les résultats au premier ordre (qui sont ceux auxquels ont s'intéresse ici) ne sont a priori pas modifiés par cette hypothèse simplificatrice.

Jusqu'au seuil de troncature qu'on note b_n, la nouvelle distribution L_1 est proportionnelle à la distribution précédente. Cette valeur seuil est définie comme le plus petit entier supérieur à $\tilde{b}_n$, avec

$$\left(1 - \frac{1}{2^{\tilde{b}_n}}\right)^{n/m} = \delta,$$

soit encore,

$$b_n = \left\lceil \log_2 \frac{1}{1 - \delta^{m/n}} \right\rceil.$$

Si M^s (s comme super) est une variable aléatoire qui suit la distribution L_1, on a

$$\Pr(M^s = k) = \frac{1}{\delta}\left(1 - \frac{1}{2^k}\right)^{n/m} - \frac{1}{\delta}\left(1 - \frac{1}{2^{k-1}}\right)^{n/m}, \ k < b_n$$

avec $\Pr(M^s = b_n)$ qui est le complément des probabilités précédentes, c'est-à-dire

$$1 - \frac{1}{\delta}\left(1 - \frac{1}{2^{b_n - 1}}\right)^{n}.$$

On définit la série génératrice de cette distribution de probabilité comme suit

$$G_s(u) := \sum \Pr(M^s = k) u^k.$$

Ici, il n'y a que la variable u qui intervient car les éléments sont par l'hypothèse H1 équirépartis dans les urnes, de sorte que n est donc traité comme un paramètre.

On peut reprendre toute la partie mise en équations formelle de l'analyse de Loglog. L'espérance notée E_s de l'estimateur $\tilde{\alpha}_m m 2^{\sum M_i/m'}$ s'exprime comme

$$E_s = \tilde{\alpha}_m m \sum_k 2^{k/m'} \Pr(\sum M_i = k) \tag{5.6}$$

Le résultat suivante reformule cette expression en fonction de la série génératrice G_s.

Résultat 5.2 *L'espérance de l'estimateur de l'algorithme super Loglog s'exprime comme*

$$E_s = \tilde{\alpha}_m m G_s(2^{1/m'})^{m'}.$$

Preuve. Ce résultat est une conséquence directe de l'expression de E_s énoncée dans l'équation (5.6), car la série génératrice $G_s(u)^{m'}$ est égale à $\sum \Pr(\sum_{i\leq m'} M_i = k)u^k$. L'évaluation de cette série en $u = 2^{1/m'}$ complète la preuve.

■

5.8.2 Espérance

Cette section étudie le comportement de l'espérance E_s de l'estimation de l'algorithme super Loglog $\tilde{e}_n$. L'analyse de E_s permet de déterminer les valeurs de correction $\tilde{\alpha}_m(n)$ et de montrer que l'espérance E_s est égale à n à des oscillations d'amplitude extrêmement faible près.

Résultat 5.3 *L'espérance E_s vérifie*

$$E_s \approx n$$

asymptotiquement, lorsque n tend vers l'infini.

Remarque. Ce résultat provient principalement de la définition de la valeur de correction $\tilde{\alpha}_m(n)$, qui à des petites oscillations près, est défini pour que ce résultat soit vrai. Cette valeur dépend de n qui est inconnu, et son approximation apporte des erreurs supplémentaires.

Cette fonction $\tilde{\alpha}_m(n)$ est périodique de période $\log_2 n$, et peut être approximée par un polynôme sur une période. Le choix d'un polynôme de degré 4 entraîne une erreur bornée par 2.10^{-6}, ce qui est suffisant pour la plupart des applications. Pour améliorer cette précision, il suffit d'approcher $\tilde{\alpha}_m$ par un polynôme de degré plus élevé, ce qui permet de rendre l'erreur commise lors de cette approximation aussi petite qu'on le souhaite.

Preuve du résultat 5.3

Cette section s'intéresse principalement à l'étude de la fonction de correction. On montre que c'est une fonction périodique qui ne dépend que de m et de la valeur κ_n, cette dernière étant équivalente à $\lceil \log_2 n/m - \log_2 \log 1/\delta \rceil - \log_2(n/m)$.

La preuve du résultat 5.3 est relativement technique, et s'articule comme suit. On reprend les grandes lignes du théorème 4.1 de l'étude de l'algorithme Loglog, avec comme difficulté supplémentaire, le fait qu'on étudie une distribution finie et non plus infinie. La preuve ramène tout d'abord l'étude de E_s à l'étude de sa racine m-ième (à une constante près) A_s, qui est l'équivalent de A dans l'analyse de l'algorithme Loglog. Cette somme A_s est décomposée en trois parties, qui sont chacune étudiées, et finalement une dernière partie rassemble tous ces résultats et conclut.

On s'intéresse à la fonction

$$A_s := G_s(2^{1/m'}) = \sum_k \Pr(M^s = k)2^{k/m'}.$$

La somme sur k dans l'équation précédente est sur le domaine $k \leq b_n$, car la distribution L_1 est (par construction) tronquée. Cela ne permet pas d'utiliser les mêmes théorèmes que dans le cas non tronqué, car la somme est finie, mais non bornée. On choisit alors de récrire cette somme comme $\sum_k = \sum_{k=0}^{\infty} - \sum_{k\geq b_n} + \sum_{k=b_n}$ car il est plus facile de traiter la somme de b_n à l'infini que son

complément, et que $k = b_n$ est un cas particulier comme on le voit sur la figure, et grâce à la définition. Pour donner un sens à ce découpage on prolonge la distribution L_1 de la façon suivante. On définit $p_k = \frac{1}{\delta}\left(1 - \frac{1}{2^k}\right)^{n/m} - \frac{1}{\delta}\left(1 - \frac{1}{2^{k-1}}\right)^{n/m}$ pour tout k, La valeur A_s s'écrit alors

$$A_s = \sum_{k=0}^{\infty} pk2^{k/m'} - \sum_{k \geq b_n} pk2^{k/m'} + \sum_n \Pr(M^s = b_n)2^{b_n/m'}.$$

La première somme, de 0 à l'infini est une reprise quasi immédiate du cas Loglog, aux modifications près que m se transforme en m', et du coefficient $1/\delta$. La deuxième somme fait apparaître des non-linéarités sous la forme d'oscillations de période $\log n$, ce qui est également le cas de la troisième somme, qui est réduite à un seul élément.

La première somme $[0, \infty]$**.** Ce paragraphe s'inspire directement de la section 4.5.4, car la première somme n'est rien d'autre que la distribution non tronquée multipliée par $1/\delta$. La première somme s'écrit

$$\frac{1}{\delta}\sum_{k \geq 0} 2^{k/m'}((1 - 2^{-k})^{n/m} - (1 - 2^{-k+1})^{n/m}).$$

Étant donné la forme de la loi de probabilité des p_k, on voit qu'on peut négliger les termes où k est petit, et donc écrire $(1 - 2^{-k})^{n/m} \sim e^{n/(m2^k)}$

La somme devient alors

$$\frac{1}{\delta}\sum_{k \geq 0} 2^{k/m'}\left(e^{-n/(m2^k)} - e^{-n/(m2^{k-1})}\right),$$

ce qui se mellinise très bien. On sait estimer cette somme en reprenant les résultats de la section 4.5.4, ce qui donne

$$\frac{1}{\delta}\sum_{k \geq 0} 2^{k/m'}\left(e^{-\frac{n'}{m'2^k}} - e^{-\frac{n'}{m'2^{k-1}}}\right) = Kn^{1/m'}$$

où K est quasiment une constante (aux petites oscillations près), définie par

$$K := \frac{1}{\delta}\Gamma(-1/m')\frac{1 - 2^{1/m'}}{\log 2}\frac{1}{m^{1/m'}}$$

Les petites oscillations mentionnées précédemment sont issues des singularités complexes de la transformée de Mellin, et ne sont pas explicitées ici. Cependant, on peut préciser que l'amplitude de ces oscillations est bornée par 10^{-6}, exactement comme dans l'analyse de Loglog.

La deuxième somme $[b_n, \infty]$ La deuxième somme, qui correspond aux valeurs tronquées y compris la valeur $k = b_n$ s'exprime sous la forme d'une somme infinie, qui est légèrement périodique en fonction de n, le résultat.

$$B(n) := \frac{1}{\delta}\sum_{k \geq b_n} 2^{k/m'}((1 - 2^{-k})^{n/m} - (1 - 2^{-k+1})^{n/m}).$$

soit encore, grâce à l'approximation $(1 - 2^{-k})^{n/m} \sim e^{-\frac{n}{m2^k}}$ lorsque k est suffisamment grand, qui s'applique ici car $k \geq b_n$

$$B(n) \sim \frac{1}{\delta}\sum_{k \geq b_n} 2^{k/m'}\left(e^{-\frac{n}{m2^k}} - e^{-\frac{n}{m2^{k-1}}}\right)$$

Pour faire apparaître les oscillations de période $\log_2 n$, on pose $b_n = \log_2(n/m) + \kappa_n$. Cela permet d'extraire le terme $n^{1/m'}$, et de faire apparaître clairement son coefficient multiplicateur, qui est une fonction de κ_n. Ce terme κ_n dépend directement de la partie fractionnaire de $\log_2 n$ lorsque n est suffisamment grand car

$$\kappa_n \sim \lceil \log_2(n/m) - \log_2 \log(1/\delta) \rceil - \log_2(n/m)$$

asymptotiquement, lorsque n tend vers l'infini.

$$B(n) \sim \frac{1}{\delta} 2^{b_n/m'} \sum_{k\geq 0} 2^{k/m'} \left(e^{-\frac{n}{m2^{k+b_n}}} - e^{-\frac{n}{m2^{k-1+b_n}}} \right)$$

$$B(n) \sim \frac{1}{\delta} (n/m)^{1/m'} 2^{\kappa_n/m'} \sum_{k\geq 0} 2^{k/m'} \left(e^{-\frac{1}{2^{k+\kappa_n}}} - e^{-\frac{1}{2^{k-1+\kappa_n}}} \right) = \beta_n n^{1/m'}$$

avec la constante β_n qui est définie par

$$\beta_n := \frac{1}{\delta} \frac{1}{m^{1/m'}} 2^{\kappa_n/m'} \sum_{k} 2^{k/m'} \left(e^{-\frac{1}{2^{k+\kappa_n}}} - e^{-\frac{1}{2^{k-1+\kappa_n}}} \right)$$

La partie β_n ne dépend en fait que de la partie fractionnaire de $\log_2 n$, et peut donc être étudiée numériquement sur un intervalle borné. Pour cela on observe que la somme présente dans l'expression de β_n converge de façon géométrique car le terme $2^{k/m'} \left(e^{-\frac{1}{2^{k+\kappa_n}}} - e^{-\frac{1}{2^{k-1+\kappa_n}}} \right)$ est majoré par $2^{k/m'} 2^{-k-\kappa_n}$. Les premiers termes de cette somme fournissent donc une approximation de la somme complète avec autant de précision qu'on souhaite, selon le nombre de termes pris en compte.

Le petit morceau $[b_n]$. La dernière partie de la somme est la partie qui correspond au fait qu'on traite une loi discrète et non continue, et qu'il y a un morceau incomplet, qu'on doit traiter séparément.

$$2^{b_n/m'} \Pr(M^s = b_n) = 2^{b_n/m'} \left(1 - \frac{1}{\delta}(1 - 2^{b_n - 1}) \right)$$

qui se récrit

$$\left(1 - \frac{\exp(-1/2^{\kappa_n - 1})}{\delta} \right) (n/m)^{1/m'} 2^{\kappa_n/m'} = \delta_n n^{1/m'}.$$

avec

$$\delta_n := \left(1 - \frac{\exp(-1/2^{\kappa_n - 1})}{\delta} \right) (1/m)^{1/m'} 2^{\kappa/m'}$$

Calculs numériques.

Finalement le total de ces trois contributions est

$$A_s = n^{1/m'}(K - \beta_n + \delta_n) = (1/m)^{1/m'} a_n n^{1/m'},$$

d'où

$$\begin{aligned} a_n \;:=\; & \frac{1}{\delta} \Gamma\left(-\frac{1}{m'}\right) \frac{1 - 2^{1/m'}}{\log 2} - \frac{1}{\delta} 2^{\kappa_n/m'} \sum_{k} 2^{k/m'} \left(e^{-\frac{1}{2^{k+\kappa_n}}} - e^{-\frac{1}{2^{k-1+\kappa_n}}} \right) \\ & + \left(1 - \frac{e^{-1/2^{\kappa_n - 1}}}{\delta} \right) 2^{\kappa_n/m'}. \end{aligned}$$

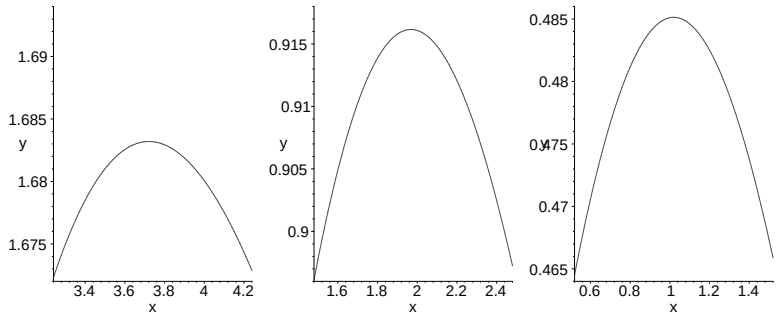

FIG. 5.20 – Le tracé de $a_n^m(\kappa_n)$ pour $\delta = 0.9$ (i), 0.7 (ii) et 0.5 (iii).

Ce qui correspond bien au Résultat 5.1. Dans toute cette partie de résultats numériques, on suppose n grand, ce qui nous permet d'approximer b_n par $\lceil \log_2(n/m) - \log_2 \log 1/\delta \rceil$. Le coefficient a_n ne dépend alors que de la partie fractionnaire de $\log_2(n/m)$ et de m. On étudie a_n numériquement dans cette section, et on rappelle que l'espérance de l'estimateur est défini comme $E_s := \tilde{\alpha}_m(n) a_n^m n$.

On constate tout d'abord, expérimentalement que les valeurs de a_n dépendent de m, mais que lorsque la partie fractionnaire de $\log_2(n/m)$ reste constante, les valeurs de a_n^m ne dépendent que faiblement de m (ce qui était déjà le cas pour l'algorithme Loglog). Ceci permet d'étudier a_n comme une fonction d'une seule variable. On choisit alors comme variable de référence κ_n, définie précédemment comme

$$\kappa_n = \left\lceil \log_2 \frac{1}{1-\delta^{m/n}} \right\rceil - \log_2(n/m) \sim \lceil \log_2 n/m - \log_2 \log 1/\delta \rceil - \log_2(n/m)$$

dont la variation est liée aux variations de la partie fractionnaire de $\log_2 n/m$, et qui varie entre $-\log_2 \log 1/\delta$ et $-\log_2 \log 1/\delta + 1$.

La figure 5.20 représente différents tracés de $a_n^m(\kappa_n)$ selon la valeur de δ. On constate sur cette figure, que selon la partie fractionnaire de $\log_2 n$, la constante de correction varie. On observe de plus que plus δ est petit, plus cette variation est grande. Cette variation pose a priori un problème, car elle ne permet pas d'avoir un algorithme donnant une estimation centrée. Il est cependant possible de régler ce problème de la façon suivante. On peut obtenir, grâce à l'algorithme Loglog une estimation de n notée n_0. Cette première estimation permet d'évaluer précisément la valeur de $\log_2 n$, et en particulier de sa partie fractionnaire. Grâce à cette connaissance, on peut savoir quelle est la constante de correction adaptée pour l'algorithme super Loglog, et ainsi obtenir un estimateur centré pour l'algorithme super Loglog.

À la place de la constante de correction α_m de l'algorithme Loglog, on pose ici une fonction de correction $\tilde{\alpha}_m(n)$, définie par

$$\tilde{\alpha}_m(n) = a_n^{-m'}$$

Fonction de correction et calcul de l'erreur commise

Pour la simplicité de l'algorithme super Loglog, on choisit d'approcher la fonction $\tilde{\alpha}_m(n)$ trouvée dans la sous section précédente par un polynôme noté P. Ce polynôme dépend de la valeur de δ et de m, et peut donc être précalculé. La figure 5.21 présente le tracé du facteur de correction $\tilde{\alpha}_m$ (qui est l'inverse du tracé de la figure 5.20), ainsi que le tracé de $\tilde{\alpha}_m - P$, qui est l'erreur commise par l'approximation polynomiale.

Il y a donc a priori deux erreurs commises sur l'évaluation de cette valeur de correction. La première provient de la différence entre $\tilde{\alpha}_m(n)$ qui est la bonne correction, et d'autre part $\tilde{\alpha}_m(n_0)$

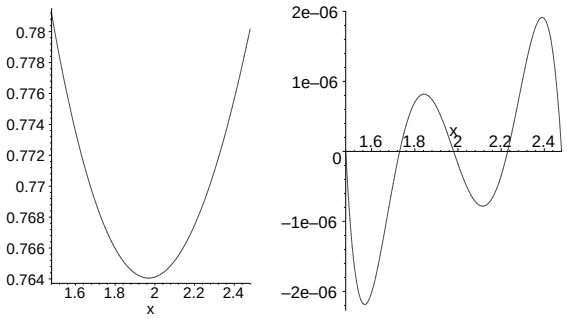

FIG. 5.21 – Le tracé de la fonction de correction $\tilde{\alpha}_m$ pour $\delta = 0.7$ avec $m = 4000$, et le tracé de $\tilde{\alpha}_m - P$, pour les mêmes valeurs.

qui est la valeur de correction pratique sur laquelle on se base. La deuxième erreur provient de l'approximation polynomiale de $\tilde{\alpha}_m(n_0)$ par $P(n_0)$.

◇ La première erreur est bornée comme suit

$$|\tilde{\alpha}_m(n) - \tilde{\alpha}_m(n_0)| \leq \text{Max}\,|\tilde{\alpha}'_m(x)||\log_2 n - \log_2 n_0|. \tag{5.7}$$

La valeur $\text{Max}(\tilde{\alpha}'_m(x))$ est une constante relativement petite (inférieure à 0.08 pour $\delta = 0.7$), et $|\log_2 n - \log_2 n_0|$ tend vers 0 lorsque m tend vers l'infini. Cette erreur est donc asymptotiquement négligeable.

Pour des valeurs de m utilisées en pratique, l'erreur créée par cette approximation doit néanmoins être prise en compte. La majoration de cette erreur obtenue dans l'équation (5.7) peut être précisée, car on connaît déjà l'erreur commise sur la première estimation n_0 issue de l'algorithme Loglog. L'erreur moyenne est obtenue en calculant l'espérance de l'équation (5.7). On cherche donc à majorer la quantité $\mathbb{E}(|\log n - \log n_0|)$ qui peut se récrire $\mathbb{E}(|\log(1 + (n_0/n - 1))|)$. Or l'estimation n_0 est proche de n, donc la quantité $n_0/n - 1$ est proche de zéro, ce qui permet de développer le logarithme en utilisant la formule $\log 1 + x \sim_0 x$. On obtient alors

$$\mathbb{E}(|\log(1 + (n_0/n - 1))|) \sim \mathbb{E}(|\frac{n_0}{n} - 1|).$$

En remarquant qu'une variance est toujours positive, on a l'inégalité $\mathbb{E}(|X|) \leq \sqrt{\mathbb{E}(X^2)}$ pour X une variable aléatoire quelconque. Cela permet d'écrire, pour n suffisamment grand

$$\mathbb{E}(|\log(1 + (n_0/n - 1))|) \leq \sqrt{\mathbb{E}}((\frac{n_0}{n} - 1)^2).$$

Le majorant de l'équation précédente est en fait exactement l'erreur standard de l'algorithme Loglog, qui est majoré par $\frac{1.3}{\sqrt{m}}$.

Finalement, l'erreur commise est majorée par $\text{Max}\,|\tilde{\alpha}'_m(x)||\log n - \log n_0|$, qui dans le cas particulier $\delta = 0.7$ on a

$$Erreur \leq \frac{1.3}{\sqrt{m}}.0.08 \leq \frac{0.11}{\sqrt{m}}.$$

Cette approximation crée une erreur qui est plus petite que l'erreur due à l'algorithme super Loglog, mais apporte tout de même une contribution d'environ 10%.

◇ La deuxième erreur est bornée par

$$|P(n_0) - \tilde{\alpha}_m(n_0)| \leq \text{Max}\,|P(x) - \tilde{\alpha}_m(x)|.$$

Cette erreur dépend de la qualité du polynôme P, et peut être rendue aussi petite qu'on veut par le choix approprié d'un polynôme de degré suffisamment grand. Pour le cas particulier $\delta = 0.7$ et $m = 4000$, en prenant un polynôme de degré 4, qui interpole $\tilde{\alpha}_m$ en 5 points, la figure 5.21 montre que cette différence est bornée par 2.10^{-6}.

Cela conclut la preuve du résultat 5.3. La section suivante estime la variance de cet estimateur, ce qui permet d'estimer la précision de ce nouvel algorithme.

5.8.3 Variance

Résultat 5.4 *L'erreur standard de l'algorithme super Loglog basé sur une valeur de correction parfaite est majorée par* $\frac{0.84}{\sqrt{m}}$, *lorsque* $\delta = 0.7$.

L'erreur commise en pratique est plus élevée, car il faut prendre en compte l'erreur commise lors du choix de la constante de correction $\tilde{\alpha}_m$ qui est majorée par $0.11/\sqrt{m}$.

Preuve–Courbes

La variance, comme l'espérance dépend de la partie fractionnaire de $\log_2 n$, mais contrairement à l'espérance, elle dépend de m. Comme pour l'algorithme Loglog, on observe que cette dépendance est en $1/m$. Ce qui intuitivement dit que plus on fait de tirage selon la distribution, plus la moyenne de ces tirages est centrée.

Toute la sous-section précédente sur l'espérance peut être adaptée pour la variance, qui est définie par

$$V_s := m^2\tilde{\alpha}_m^2(G_s(2^{2/m'})^{m'} - G_s(2^{1/m'})^{2m'}).$$

On étudie alors la valeur M_2, définie par

$$M_2 := G_s(2^{2/m'}) = \sum_k \Pr(M^s = k)2^{2k/m'}.$$

Les calculs sont similaires à ceux de l'espérance, et ne sont pas repris. On obtient

$$M_2 \sim v_n n^{2/m'}$$

lorsque n tend vers l'infini, v_n dépend de la partie fractionnaire de $\log_2 n/m$ et également de m, et en reprenant les notations de la section précédente est égale à

$$v_n := \frac{m^{-2/m'}}{\delta}\Gamma\left(\frac{-2}{m'}\right)\frac{1-2^{2/m'}}{\log(2)} + \left(1 - \frac{e^{-1/2^{\kappa_n-1}}}{\delta}\right)m^{-2/m'}2^{2\kappa_n/m'}$$
$$- \frac{1}{\delta}(1/m)^{2/m'}2^{2\kappa_n/m'}\sum_k 2^{2k/m'}\left(e^{-1/2^{k+\kappa_n}} - e^{-1/2^{k+\kappa_n-1}}\right).$$

La variance est ensuite définie comme $V_s := \tilde{\alpha}_m^2 m^2 v_n^{m'} - E_s^2$, et l'erreur standard par $err_s := \frac{\sqrt{V_s}}{E_s}$. Ce qui nous intéresse maintenant est de pouvoir borner l'erreur standard, selon les différentes valeurs de δ. La figure 5.22 représente des tracés de la variance $V_s n^{-2}$ pour différentes valeurs de δ, en fonction de κ_n. Le tableau suivant (5.8) représente les valeurs maximales de l'erreur standard, rapportées à m, pour différentes valeurs de δ.

δ	0.9	0.8	0.7	0.6	0.55	0.5	0.45	0.4
Err stand $\times \sqrt{m}$	0.98	0.88	0.84	0.797	0.790	0.787	0.789	0.796

(5.8)

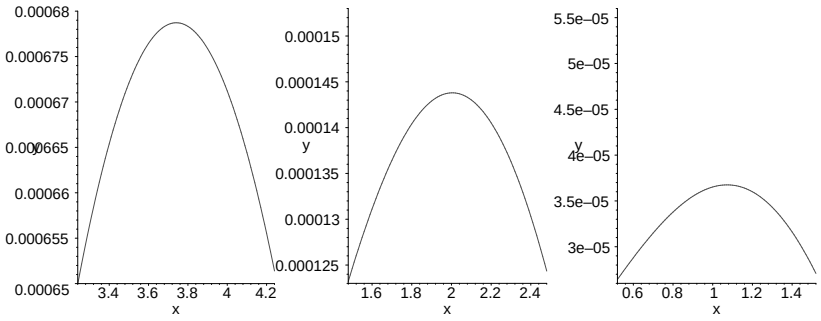

FIG. 5.22 – Le tracé de la constante de la variance pour $\delta = 0.9$ (i), 0.7 (ii) et 0.5 (iii), avec $m = 4000$.

On constate alors que la troncature optimale, selon les critères vus jusqu'ici, est située à $\delta = 0.5$, et que l'erreur standard est alors $\frac{0.79}{\sqrt{m}}$. La section suivante est consacrée au choix de la valeur de troncature.

5.8.4 Conclusion

Choix du taux de troncature. On observe que l'analyse suggère un taux de troncature égal à 0.5, alors que l'expérience conduit à un taux de troncature voisin de 0.7. En pratique il est raisonnable de se fier à la valeur expérimentale, qui se base sur des valeurs de m finies. De plus l'analyse montre que l'erreur standard varie peu (quelques %) lorsque δ varie entre 0.4 et 0.7.

Erreur globale. Une fois le taux de troncature choisi égal à 0.7, l'erreur standard est majorée par la somme de l'erreur standard de l'algorithme doté d'un oracle qui lui fournit le facteur de correction et de l'erreur due à l'approximation de $\tilde{\alpha}_m$. L'erreur due à l'approximation polynomiale de $\tilde{\alpha}_m$ est en pratique négligeable et on obtient alors que asymptotiquement

$$\text{Erreur moyenne} \leq \frac{0.95}{\sqrt{m}}.$$

Cette valeur est légèrement optimiste par rapport aux valeurs observées pour des m petits (on trouve $1.04/\sqrt{m}$ pour $m = 2^8$), mais devient valide lorsque m augmente ($0.94/\sqrt{m}$ pour $m = 2^{20}$).

5.9 Code source

Les algorithmes Loglog et super Loglog sont présentés en pseudo code dans les sections 4.4.3 et 5.2.2. Dans cette section, on présente le code complet de ces deux algorithmes écrit en C. Ce code est basé sur une première implémentation aimablement fournie par Keith Briggs.

Les différentes parties du code sont séparées deux sections. La première contient le corps du programme et les fonctions essentielles comme Loglog, super Loglog, le hachage et la correction des non-linéarités. La deuxième contient les fonctions moins importantes, qui sont nécessaires pour faire tourner le programme, mais ne sont pas indispensables pour la lisibilité et la compréhension du code.

5.9.1 Fonctions principales

Le code des principales fonctions est séparé en plusieurs paragraphes. Le premier concerne les notations, et le second les fonctions de hachages, qui sont nécessaires pour transformer les données

en mots binaires de manière satisfaisante. Le troisième et le quatrième présentent les algorithmes Loglog et Super Loglog, et enfin le dernier la fonction qui permet de corriger les non linéarités.

En tête, notations

Le programme se sert des librairies standard suivantes :

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
```

Les variables globales sont :

```
typedef unsigned int ui;
ui k;           //taille du prefixe
ui m;           //nombre de registres
double alpha;   // constante de correction
int stop=0;
```

La fonction de hachage

La fonction de hachage est en deux parties, tout d'abord convertir la chaîne de caractères en entier, puis hacher cet entier.

```
ui convert(char* t){//convertit une string en ui
  ui res=0; int pow=1;int i;
  for(i=0;i<strlen(t);i=i+1) {
    res+=pow*(int)t[i];
    pow=(pow<<5)+(pow<<2)+pow;}
  return res;}
```

Cette fonction de hachage est extraite de [Knu98, section 6.4]. La valeur `K` doit être un grand nombre premier. En utilisant plusieurs valeurs de `K` on peut obtenir un grand nombre de fonctions de hachage différentes.

```
ui hash(int a){
   double K=179424673.0;
   double v0=((double)a/pow(2,32))*K;
   return (ui)((double)pow(2,32)*(v0-(int)v0));}
```

La fonction Loglog

Les constantes de correction α sont précalculées. Lorsque le nombre de registres est supérieur à 9, on peut considérer que α_m ne dépend plus de m et est égal à 0.3965.

```
double loglog(int* M){
  double b=0; int i; double alpha=0.3965;
  switch (k){
  case 4:alpha=0.376;break;case 5:alpha=0.387;break;case 6:alpha=0.391;break;
  case 7:alpha=0.394;break;case 8:alpha=0.396;break;case 9:alpha=0.396;break;
  }
```

```
  for(i=0;i<m;i++){b+=M[i];}
  return pow(2,b/m)*(alpha)*m;
}
```

L'algorithme Super Loglog

Cette première méthode `alphanum` calcule la constante de correction α, en fonction d'une première évaluation `nval` et de la valeur de `k`. A chaque `k` est associé un polynôme qui approxime la véritable fonction de correction. Ces polynômes deviennent très proches lorsque `k` devient grand, ce qui permet de regrouper certains cas. Les valeurs de `k` inférieures à 3 ne sont pas traitées ici, mais peuvent l'être sans difficulté supplémentaire.

```
double alphanum(double nval){
  double kap=(int)(log(nval/m)/log(2)+1.48)+1-log(nval/m)/log(2);
  if(k<4){return 0.74;}
  switch (k){
    case 4:alpha=0.003497*pow(kap,4)-0.03555*pow(kap,3)+0.1999*pow(kap,2)
                -0.4812*kap+1.1390;break;
    case 5:alpha=0.00324250*pow(kap,4)-0.0346687*pow(kap,3)+1.140732+
                0.19794194*pow(kap,2)-0.47555735320*kap;break;
    case 6:alpha=0.0031390489*pow(kap,4)-0.0343776755*pow(kap,3)+
                1.141759+0.197295*pow(kap,2)-0.4730536*kap;break;
    case 7:alpha=0.0030924632*pow(kap,4)-0.0342657653*pow(kap,3)+
                1.142318+0.197045*pow(kap,2)-0.4718622*kap;break;
    case 8:alpha=0.0030709*pow(kap,4)-0.034219*pow(kap,3)+1.14260+
                0.19694*pow(kap,2)-0.47129*kap;break;
    case 9: case 10: case 11:case 12 : alpha=0.0030517*pow(kap,4)-
            0.034180*pow(kap,3)+1.14287+0.19685*pow(kap,2)-0.47077*kap;break;
    default :alpha=0.0030504*pow(kap,4)-0.034177*pow(kap,3)+1.14288+
                  0.19685*pow(kap,2)-0.47073*kap;break;
    }
  return alpha;
}

double superloglog(int M[]){
  double ave=0.0;
  alpha=alphanum(loglog(M));
  int prop=m*7/10;  // La valeur de troncature choisie ici (70%)
  ave=sommetronc(M,prop)*1.0/prop;
  return alpha*m*pow(2,ave);
}
```

La correction des non-linéarités

Cette méthode met en place une estimation du nombre d'éléments par comptage de collision, présenté dans la section 5.3. Cette méthode est ensuite appelée par la méthode `evaluation`.

```
double nonlincc(int M[]){
  int nb=0;int i;
```

```
  for (i=0;i<m;i++){if(M[i]==0){nb++;}}
  if(nb==0){nb=1;}
  if((m-nb)<sqrt(m)){return (double)(m-nb);}
  return m*log(m/(double)nb);
}
```

La fonction principale

La méthode `evaluation` rassemble tous les éléments précédents, et décide quel valeur utiliser entre le résultat de `superloglog` et le résultat de `nonlincc`. Enfin cette méthode corrige l'effet des collisions lors du hachage.

```
double evaluation(int M[]){
  double res;
  double sll=superloglog(M);
  double nl=nonlincc(M);
  if(sll<5*m/2 && nl<5*m/2){ res=nl;}
  else {if(sll>5*m/2 && nl>5*m/2){ res=sll;}
  else res= (sll+nl)/2;}

  float N=pow(2.,32); //Correction des collisions
  return (-N*log(1-res/N));
}
```

La fonction principale `main` initialise toutes les variables nécessaires, procède à la mise à jour des registres, et enfin affiche le résultat souhaité.

```
int main(int argc, char* argv[]) {
  ui b,j,*M,r;double card;FILE *fich;
  if(argc<2){
    printf("To run the algorithm, run the command\n
    ./superloglog m filename\n m should be an integer<32,  and filename the
    name of the file.\n
    The result returned is an estimate of the number of distinct words of
    filename\n");
    return 1;
  }
  k=atoi(argv[1]); k=k<1?1:k; k=k>32?32:k;
  // Verifie que la valeur de k est acceptable
  m=1<<k;

  fich=fopen(argv[2],"r");M=calloc(m,sizeof(ui));
  b=sourcefich(fich);
  while (stop==0) {
    j=head(b);r=rho(tail(b));if (r>M[j]) M[j]=r;b=sourcefich(fich);
  }
  card=evaluation(M);
  fprintf(stderr,"The number of distinct words in the file %s is %f,
  with an expected precision of %f pc\n",argv[2],card, 0.95/sqrt(m)*100);
```

```
  free(M);
  return 0;
}
```

5.9.2 Fonctions annexes

```
int sommetronc(int M[],int lim){//somme les  70% plus petites valeurs de M
  int *C;int i;int ct=0;int pos=0;int res=0;
  C=calloc(32-k+2,sizeof(int));
  for(i=0;i<m;i++){C[M[i]]++;}
  while(ct+C[pos]<=lim){
    res+=pos*C[pos];ct+=C[pos];pos++;
  }
  res+=(lim-ct)*pos;
  free(C);
  return res;
}

ui rho(int x) { // index du premier 1 dans x, en comptant à partie de 1
  int i;
  for (i=1; i<=32; i++) {if (x<0) return i;x<<=1;}
  return 33-k;
}

ui head(ui x) { //  k premiers bits de x
  return x>>(32-k);}

ui tail(ui x) { // supprime les k premiers bits de  x
  return x<<k;}

ui sourcefich(FILE* file){//lit le fichier ``file'' mot à mot
  char toto[2000];
  if(file==NULL){
    printf("Error: can't open file.\n");
    stop=1;return 0;
  }
  else {if(!feof(file)) {
      fscanf(file, "%s", &toto);}
    else {stop=1;return 0;}
  }
  ui a=hash(convert(toto));
  return a;
}
```

Conclusion

L'application de la combinatoire analytique à l'analyse d'algorithmes a permis dans cette thèse d'obtenir de nombreux résultats. Cette approche permet de traduire les propriétés de décomposabilité des algorithmes ou des structures en séries génératrices. Cela permet d'exprimer de manière complète le comportement de certains paramètres sur des structures parfois complexes. La mise en équation au chapitre 3 du coût de construction d'une table de hachage par l'algorithme de hachage à essai aléatoire en est un excellent exemple. Les résultats asymptotiques s'obtiennent alors de manière simple comme une évaluation d'une série génératrice ou de ses dérivées. On voit dans les chapitres 2, 4 et 5 que cette approche permet également d'atteindre des résultats asymptotiques a priori complexes. Par exemple le comportement en $\log k + \log n - k$ de la profondeur des éléments dans une liste à sauts de taille n, ou le comportement très légèrement oscillant des estimateurs des algorithmes Loglog et super Loglog. Ces exemples confirment le fait que la combinatoire analytique est une approche élégante et puissante à l'étude des problèmes de complexité algorithmique.

Les résultats analytiques bivariés développés au chapitre 2 concernent une large classe de problèmes. On a vu les listes à sauts, les algorithmes de type quicksort ainsi qu'une estimation de fréquence d'apparition de symboles.

Lors de l'analyse de l'algorithme de hachage à essai aléatoire, on a obtenu la série génératrice multivariée qui code le coût de construction et la répartition des tables de hachage. Cette série génératrice a mené à l'estimation de l'espérance et de la variance du coût de construction pour les tables α-pleines et presque pleines. Beaucoup d'autres problèmes peuvent être analysés en se basant sur cette série génératrice. Par exemple la manière dont se passe, pour le coût de construction, la transition entre les tables α-pleines et presque pleines, ou encore l'évolution de la répartition des tables de hachage selon le taux de remplissage. Il serait intéressant de déterminer si les méthodes utilisées pour le hachage à essai aléatoire s'appliquent à des stratégies de hachage voisins, comme le hachage à essai uniforme.

Les algorithmes de comptage probabiliste présentés aux chapitres 4 et 5 sont actuellement les meilleurs algorithmes d'estimation de cardinalité, domaine très actif grâce à la motivation apportée par les routeurs. Ceci illustre à mon sens la force de l'analyse dans la conception algorithmique. D'autres algorithmes, également basés sur la répartition de valeurs aléatoires (qui représentent les données hachées) sont envisageables, et l'analyse générale de ces algorithmes constitue un projet de recherche prometteur.

Bibliographie

[AS73] Milton ABRAMOWITZ et Irene A. STEGUN. *Handbook of Mathematical Functions*. Dover, 1973. A reprint of the tenth National Bureau of Standards edition, 1964.

[AMS99] Noga ALON, Yossi MATIAS, et Mario SZEGEDY. The space complexity of approximating the frequency moments. *J. Comp. Sys. Sci.*, 58 :137–147, 1999.

[And91] Arne ANDERSSON. A note on searching in a binary search tree. *Software—Practice and Experience*, 21(10) :1125–1128, 1991.

[ASW87] M. ASTRAHAN, M. SCHKOLNICK, et K. WHANG. Approximating the number of unique values of an attribute without sorting. *Information Systems*, 12(1) :11–15, 1987.

[BM93] Jon L. BENTLEY et Douglas MCILROY. Engineering a sort function. *Software—Practice and Experience*, 23(11) :1249–1265, 1993.

[BCGL03] Alberto BERTONI, Christian CHOFFRUT, Massimiliano GOLDWURM, et Violetta LONATI. On the number of occurrences of a symbol in words of regular languages. *Theoretical Computer Science*, 302(1-3) :431–456, 2003.

[BCD03] Hervé BRÖNNIMANN, Frédéric CAZALS, et Marianne DURAND. Randomized jumplists : A jump-and-walk dictionary datastructure. Dans *Proceedings of the STACS'03 Conference, Berlin, February 2003*, volume 2607 de *Lecture Notes in Computer Science*, pages 283–294. Springer Verlag, 2003.

[CDJH01] Brigitte CHAUVIN, Michael DRMOTA, et Jean JABBOUR-HATTAB. The profile of binary search trees. *Ann. Appl. Prob.*, 11 :1042–1062, 2001.

[Chy98] Frédéric CHYZAK. *Fonctions holonomes en calcul formel*. Thèse universitaire, École polytechnique, 1998. INRIA, TU 0531. 227 pages.

[Com74] Louis COMTET. *Advanced Combinatorics*. Reidel, Dordrecht, 1974.

[CGHJ93] Robert M. CORLESS, Gaston H. GONNET, D. E. G. HARE, et David J. JEFFREY. Lambert's W function in Maple. *Maple Technical Newsletter*, 0(9) :12–22, Spring 1993.

[CLRS01] Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, et Clifford STEIN. *Introduction to Algorithms*. MIT Press and McGraw-Hill, seconde édition, 2001.

[DGL03] Diego DE FALCO, Massimiliano GOLDWURM, et Violetta LONATI. Frequency of symbol occurences in bicomponent stochastic models. *Rapporto Interno n. 296-03, Dip. Scienze dell'Informazione, Università degli Studi di Milano, Giugno*, 2003.

[Dev86] Luc DEVROYE. A note on the height of binary search trees. *Journal of the ACM*, pages 489–498, 1986.

[Dev87] Luc DEVROYE. Branching processes in the analysis of the heights of trees. *Acta Informatica*, 24 :277–298, 1987.

[Drm94a] Michael DRMOTA. Asymptotic distributions and a multivariate darboux method in enumeration problem. *J. Combinatorial Theory, serie A*, 67 :169–184, 1994.

[Drm94b] Michael DRMOTA. A bivariate asymptotic expansion of coefficients of powers of generating functions. *European Journal of Combinatorics*, 15 :139–152, 1994.

[Drm94c] Michael DRMOTA. The height distribution of leaves in rooted trees. *Discrete Mathematics and Applications*, 4 :45–58, 1994.

[Drm01] Michael DRMOTA. An analytic approach to the height of binary search trees. *Algorithmica*, 29(1) :89–119, 2001.

[DS95] Michael DRMOTA et Michèle SORIA. Marking in combinatorial constructions : Generating functions and limiting distributions. *Theoretical Computer Science*, 144(1–2) :67–99, juin 1995.

[Dur00] Marianne DURAND. Rapport de DEA. Holonomie et applications en analyse d'algorithmes et combinatoire. Disponible à l'adresse `http://algo.inria.fr/durand/`, 2000.

[Dur03] Marianne DURAND. Asymptotic analysis of an optimized quicksort algorithm. *Information Processing Letters*, 85 :73–77, 2003.

[DF03] Marianne DURAND et Philippe FLAJOLET. Loglog counting of large cardinalities. Dans G. DI BATTISTA et U. ZWICK, éditeurs, *Annual European Symposium on Algorithms (ESA03)*, volume 2832 de *Lecture Notes in Computer Science*, pages 605–617, septembre 2003.

[DT03] Marianne DURAND et Stephen TAYLOR. Emerging behavior as binary search trees are symetrically updated. *Theoretical Computer Science*, 297 :425–445, 2003.

[EV01] Cristian ESTAN et George VARGHESE. New directions in traffic measurement and accounting. Dans *ACM SIGCOMM Internet Measurement Workshop*, pages 75–80, Nov 2001.

[EV02] Cristian ESTAN et George VARGHESE. Counting active flows in high-speed routers. Preprint, 2002.

[EVF02] Cristian ESTAN, George VARGHESE, et Mike FISK. Counting the number of active flows on a high speed link. Dans *ACM SIGCOMM Computer Communication Review*, volume 32, pages 10–10, July 2002.

[EVF03] Cristian ESTAN, George VARGHESE, et Mike FISK. Bitmap algorithms for counting active flows on high speed links. Dans *Internet Measurement Conference*, mars 2003.

[Fla90] Philippe FLAJOLET. On adaptive sampling. *Computing*, 34 :391–400, 1990.

[FGP03] Philippe FLAJOLET, Joaquim GABARRÓ, et Helmut PEKARI. Analytic urns. 2003. Submitted to Annals of Probability.

[FGD95] Philippe FLAJOLET, Xavier GOURDON, et Philippe DUMAS. Mellin transforms and asymptotics : Harmonic sums. *Theoretical Computer Science*, 144(1-2) :3–58, 1995.

[FM85] Philippe FLAJOLET et G. Nigel MARTIN. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2) :182–209, 1985.

[FO90] Philippe FLAJOLET et Andrew M. ODLYZKO. Singularity analysis of generating functions. *SIAM Journal on Algebraic and Discrete Methods*, 3(2) :216–240, 1990.

[FPV98] Philippe FLAJOLET, Patricio POBLETE, et Alfredo VIOLA. On the analysis of linear probing hashing. *Algorithmica*, 22(4) :490–515, décembre 1998.

[FS05] Philippe FLAJOLET et Robert SEDGEWICK. *Analytic Combinatorics*. 2005. Livre en préparation : Les chapitres sont disponibles sous forme de rapports de recherche INRIA, numéros 1888, 2026, 2376, 2956, 3162, 4103.

[GPA+98] Phillip B. GIBBONS, Viswanath POOSALA, Swarup ACHARYA, Yair BARTAL, Yossi MATIAS, S. MUTHUKRISHNAN, Sridhar RAMASWAMY, et Torsten SUEL. AQUA : System and techniques for approximate query answering. Tech. report, Bell Laboratories, Murray Hill, New Jersey, février 1998.

[GBY91] Gaston H. GONNET et Ricardo BAEZA-YATES. *Handbook of Algorithms and Data Structures- In Pascal and C*. Addison-Wesley, seconde édition, 1991.

[GKP94] Ronald L. GRAHAM, Donald E. KNUTH, et Oren PATASHNIK. *Concrete Mathematics : A Foundation for Computer Science*. Addison Wesley, seconde édition, 1994.

[GS78] Leo GUIBAS et Endre SZEMEREDI. The analysis of double hashing. *Journal of Computer and System Sciences*, 16 :226–274, 1978.

[Hoa61] Charles A. R. HOARE. Find. *Communications of the ACM*, 4(7) :321–322, 1961.

[Hoa62] Charles A. R. HOARE. Quicksort. *Computer J.*, 5(1) :10–15, 1962.

[Hwa94] Hsien-Kuei HWANG. *Théorèmes limites pour les structures combinatoires et les fonctions arithmetiques*. Thèse de doctorat, École Polytechnique, décembre 1994.

[Jab01] Jean JABBOUR-HATTAB. Martingales and large deviations for binary search trees. *Random Structures and Algorithms*, 19(2) :112–127, 2001.

[JS98] Philippe JACQUET et Wojciech SZPANKOWSKI. Analytical depoissonization and its applications. *Fondamental Study*, 201(1-2), 1998.

[KPS92] P. KIRSCHENHOFER, H. PRODINGER, et W. SZPANKOWSKI. How to count quickly and accurately : A unified analysis of probabilistic counting and other related problems. Dans W. KUICH, éditeur, *Automata, Languages and Programming*, volume 623 de *Lecture Notes in Computer Science*, pages 211–222, 1992.

[KPS96] P. KIRSCHENHOFER, H. PRODINGER, et W. SZPANKOWSKI. Analysis of a splitting process arising in probabilistic counting and other related algorithms. *Random Structures & Algorithms*, 9 :379–401, 1996.

[Knu98] Donald E. KNUTH. *The Art of Computer Programming*, volume 3 : Sorting and Searching. Addison-Wesley, seconde édition, 1998.

[KW66] Alan G. KONHEIM et Benjamin WEISS. An occupancy discipline and applications. *SIAM Journal on Applied Mathematics*, 14 :1266–1274, 1966.

[Lar83] Per-Åke LARSON. Analysis of uniform hashing. *Journal of the ACM*, 30(4) :805–819, 1983.

[LM93] George LUEKER et Mariko MOLODOWITCH. More analysis of double hashing. *Combinatorica*, 13 :83–96, 1993.

[Mah92] Hosam MAHMOUD. *Evolution of Random Search Trees*. John Wiley, New York, 1992.

[Mar04] Conrado MARTINEZ. Partial quicksort. Dans *Proc. of the First ACM-SIAM Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, 2004.

[MPV04] Conrado MARTÍNEZ, Daniel PANARIO, et Alfredo VIOLA. Adaptive sampling for quickselect. Dans *Proc. of the 15thACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2004.

[MR01] Conrado MARTÍNEZ et Salvador ROURA. Optimal sampling strategies in quicksort and quickselect. *SIAM J. on Computing*, 31(3) :683–705, 2001.

[MR95] Rajeev MOTWANI et Prabhakar RAGHAVAN. *Randomized Algorithms*. Cambridge University Press, 1995.

[NS60] Donald J. NEWMAN et Lawrence SHEPP. The double dixie cup problem. *The American Mathematical Monthly*, 67 :58–61, 1960.

[PSF+01] Christopher R. PALMER, Georgos SIGANOS, Michalis FALOUTSOS, Christos FALOUTSOS, et Phillip GIBBONS. The connectivity and fault-tolerance of the internet topology. Dans *NRDM-2001*, 2001.

[PW97] Robin PEMANTLE et Mark WILSON. Asymptotics of multivariate sequences, part I : smooth points of the singular variety. *J. Comb. Theory, Series A*, 97 :129–161, 1997.

[PW03] Robin PEMANTLE et Mark WILSON. Asymptotics of multivariate sequences II. multiple points of the singular variety. to appear, 2003.

[PVM97] Patricio V. POBLETE, Alfredo VIOLA, et J. Ian MUNRO. The diagonal poisson transform and its application to the analysis of a hashing scheme. *Random Structures and Algorithms*, 10(1-2) :221–255, 1997.

[Pro96] Helmut PRODINGER. Combinatorics of geometrically distributed random variables : Left-to-right maxima. *Discrete Mathematics*, 153 :253–270, 1996.

[Pug90] William PUGH. Skip lists : a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6) :668–676, juin 1990.

[Puy04] Vincent PUYHAUBERT. *Ma vie, mon oeuvre*. Thèse, École polytechnique, 2004.

[Ram87] M. V. RAMAKRISHNA. Computing the probability of hash table/urn overflow. *Commun. Stat. Theor. Methods*, 16(11) :3343–3353, 1987.

[Ram88] M. V. RAMAKRISHNA. Hashing in practice, analysis of hashing and universal hashing. Dans *ACM SIGMOD international conference on Management of data*, pages 191–199, 1988.

[Ram89] M. V. RAMAKRISHNA. Analysis of random probing hashing. *Information Processing Letter*, 31(2) :83–90, 1989.

[Ree03] Bruce REED. The height of a random binary search tree. *J. ACM*, 50(3) :306–332, 2003.

[SZ94] Bruno SALVY et Paul ZIMMERMANN. Gfun : a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2) :163–177, 1994.

[Sed77] Robert SEDGEWICK. Quicksort with equal keys. *SIAM Journal on Computing*, 6(2) :240–267, juin 1977.

[Sed88] Robert SEDGEWICK. *Algorithms*. Addison–Wesley, second édition, 1988.

[Sen81] Eugene SENETA. *Non-negative matrices and Markov chains*. Springer-Verlag, New York,, second édition, 1981.

[Sta80] R. P. STANLEY. Differentiably finite power series. *European Journal of Combinatorics*, 1 :175–188, 1980.

[Sta98] R. P. STANLEY. *Enumerative Combinatorics*, volume II. Wadsworth & Brooks/Cole, 1998.

[Szp01] Wojciech SZPANKOWSKI. *Average-Case Analysis of Algorithms on Sequences*. John Wiley, New York, 2001.

[VP98] Alfredo VIOLA et Patricio POBLETE. The analysis of linear probing hashing with buckets. *Algorithmica*, 21 :37–71, 1998.

[Vui80] J. VUILLEMIN. A unifying look at data structures. *Communications of the ACM*, 23(4) :229–239, avril 1980.

[WVZT90] Kyu-Young WHANG, Brad T. VANDER-ZANDEN, et Howard M. TAYLOR. A linear-time probabilistic counting algorithm for database applications. *TODS*, 15(2) :208–229, 1990.

[Wid41] David Vernon WIDDER. *The Laplace Transform*. Princeton University Press, 1941.

Index