HAL
open science

Concurrent process and combinatorics of increasingly labeled structures : quantitative analysis and random generation algorithms

Matthieu Dien

To cite this version:

Matthieu Dien. Concurrent process and combinatorics of increasingly labeled structures : quantitative analysis and random generation algorithms. Data Structures and Algorithms [cs.DS]. Université Pierre et Marie Curie - Paris VI, 2017. English. NNT : 2017PA066210 . tel-01678893v2

HAL Id: tel-01678893

https://theses.hal.science/tel-01678893v2

Submitted on 25 Jan 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Pierre et Marie Curie
École Doctorale Informatique, Télécommunications et Électronique

Processus concurrents et combinatoire des structures croissantes : analyse quantitative et algorithmes de génération aléatoire.

Matthieu Dien
sous la direction de Michèle Soria
et l'encadrement d'Antoine Genitrini et Frédéric Peschanski

THÈSE
pour obtenir le titre de
Docteur en Sciences mention Informatique

présentée devant le jury composé de

Olivier Bodini	Université Paris 13
Alain Denise (rapporteur)	Université Paris-Sud
Antoine Genitrini	Université Pierre et Marie Curie
Conrado Martínez	Universidad Politècnica Catalunya
Antoine Miné	Université Pierre et Marie Curie
Cyril Nicaud (rapporteur)	Université Paris-Est
Frédéric Peschanski	Université Pierre et Marie Curie
Michèle Soria	Université Pierre et Marie Curie

Table des matières

Table des matières **i**

Introduction **1**
Contexte général 1
Approches alternatives 2
Contributions de la thèse 3
Plan du manuscrit 3

1 Diamants Croissants **5**
1.1 Présentation 5
1.2 Combinatoire des diamants croissants 8
1.3 Bijections 17
1.4 Comportement asymptotique 23
1.5 Degré à la racine 27
1.6 Conclusion 29

2 Ordres Partiels Série-Parallèle **31**
2.1 Introduction 31
2.2 Ordres Série-Parallèle et Graphes Fork-Join 32
2.3 Graphes Fork-Join croissants et produit ordonné 36
2.4 Graphes Fork-Joins croissants de profondeur fixée et produit coloré 42
2.5 Produits ordonné et coloré : propriétés fondamentales et exemples 45

3 Génération aléatoire **55**
3.1 Génération aléatoire uniforme de Boltzmann 55
3.2 Expérimentations : les diamants généraux 62
3.3 Extensions linéaires d'ordres Série-Parallèle 70
3.4 Conclusion 83

4 Ouverture : ordres partiels sans cycle **85**
4.1 Interprétation géométrique des ordres partiels 85
4.2 La décomposition *Bottom, Intermediate, Top and Cycle* (BITC) 86
4.3 Ordres partiels sans cycle 88
4.4 Expérimentation 91

Conclusion et Perspectives **93**

Bibliographie **97**

A Ordres partiels **103**

B Transformées de Laplace et de Borel **107**

Introduction

Contexte général

D'innombrables activités humaines sont aujourd'hui supportées par des logiciels de complexité et de taille croissante. Avec l'avènement des architectures multi-coeurs embarquées, des facteurs importants de cette complexité sont liés au parallélisme et à la concurrence.

Un programme concurrent est composé de plusieurs unités logiques : les processus. Chaque processus a un comportement qui lui est propre : il exécute ses actions de façon séquentielle. Quand plusieurs processus s'exécutent en parallèle, l'ordre d'exécution des actions du programme global n'est plus déterminé. En pratique, ce parallélisme peut être matériel (architecture multi-coeur, clusters de machines, etc.) ou logiciel (ordonnanceur de système d'exploitation, etc.).

Un objectif important est de s'assurer que de tels système concurrents complexes sont exempts de défaut. Cette problématique est étudiée depuis les années 60 dans le cadre de la théorie de la concurrence (cf. [Pet62],[Hoa78] et [Mil80]). De nombreuses techniques et méthodes d'analyse ont été proposées : *model checking*, analyse statique, tests automatisés, etc.

Ces approches diverses se rejoignent sur un problème commun connu dans le folklore comme le phénomène d'"*explosion combinatoire*". Car si l'ordre d'exécution des actions n'est pas déterminé, c'est surtout le grand nombre d'exécutions possibles qui pose problème. Cette thèse s'inscrit dans un projet à long terme d'étude quantitative de ce phénomène en exploitant notamment les outils de la combinatoire analytique [FS09]. Dans des travaux précédents ont notamment été étudiés le principe d'entrelacement [BGP16] et le non-déterminisme [BGP13].

Notre objectif est d'aborder une autre composante fondamentale de la concurrence : la synchronisation. Ce principe permet aux processus de communiquer entre eux. Par exemple, dans le cas d'un programme parallèle, plusieurs processus vont effectuer des sous-calculs indépendants avant de se synchroniser pour produire le résultat final. En pratique, la synchronisation peut être faite avec un mécanisme de mémoire partagée (mutex, sémaphore, etc.) ou en utilisant des canaux de communications (CSP, CCS, etc.).

Les travaux précédents mettent en exergue un lien très fort entre concurrence et combinatoire des structures croissantes (cf. [Gen17] pour une vue d'ensemble). Les études précédentes étaient principalement basées sur l'étiquetage croissant de structures arborescentes mais l'approche proposée dans cette thèse nécessite l'introduction des graphes dirigés acycliques (*DAG* en anglais).

De nombreux formalismes restreignent la structure de contrôle des programmes à un graphe dirigé acyclique. Des exemples illustratifs sont les langages synchrones [PEB07] et la programmation fonctionnelle réactive (*FRP* en anglais) [ACRS16]. On retrouve aussi cette contrainte en

calcul parallèle notamment dans le *Bulk Synchronous Parallel model* (*BSP*, cf. [Val90]) et en *ordonnancement* (cf. [JCB00]).

Du point de vue de la combinatoire l'étude des DAGs croissants est une problématique complexe. Un résultat fondamental établi dans [BW91] est que le problème de compter le nombres d'extensions linéaires d'un ordre partiel arbitraire (correspondant au nombre d'exécutions possibles d'un programme) est $\sharp$P-complet. Cela signifie que même pour ce modèle simplifié de concurrence, l'étude du phénomène d'explosion combinatoire est complexe.

Dans cette thèse notre approche est de considérer des sous-classes de programmes, donc de DAGs, pour lesquelles le problème de comptage semble plus accessible. Tout d'abord, nous nous intéressons à la classe des *diamants croissants* pour lesquels nous avons pu exploiter les outils de combinatoire analytique (cf. [BDF$^+$16]). Ensuite, dans l'article [BDGP17] nous avons étudié une classe de programmes correspondant aux ordres partiels *Série-Parallèle* et pour lesquels nous avons pu produire des résultats asymptotiques, mais en passant par des techniques plus ad-hoc. Enfin, nous abordons une classe beaucoup plus expressive, les ordres partiels *sans cycle*, pour laquelle nous n'avons que des résultats préliminaires (encourageants).

Si cette étude est principalement théorique, nous envisageons cependant à plus long terme des applications pratiques. Une motivation importante de notre travail concerne en effet la vérification automatique du bon fonctionnement de programmes concurrents. L'analyse exhaustive du comportement des programmes concurrents se heurte au phénomène d'explosion combinatoire. De nombreuses techniques de réduction de l'espace d'états ont été proposées : compression, abstraction, élimination des symétries, etc. Une autre approche complémentaire consiste en une vérification statistique du programme.

Pour parer à l'explosion combinatoire, une idée est d'abandonner l'exhaustivité de la vérification en adoptant une approche probabiliste. L'objectif est de s'assurer que quelques exécutions choisies au hasard sont "bonnes". Il a été observé que si le hasard est biaisé pour favoriser une petite fractions des exécutions possibles, alors les résultats de la vérification peuvent se révéler mauvais (cf. [GS05])). Au contraire, une plus grande uniformité semble donner de meilleurs résultats (cf. [ODG$^+$11]). De part les nombreux liens existants entre combinatoire et génération aléatoire, nous nous intéressons donc aussi au problème de la génération aléatoire uniforme d'exécutions. Enfin, une dernière motivation, pouvant s'appliquer à la concurrence mais plus large dans les faits, concerne la génération aléatoire uniforme de programmes. De même que générer aléatoirement des exécutions possibles d'un programme pour le vérifier, on peut générer aléatoirement un programme pour vérifier que les outils travaillant sur les programmes (compilateurs, analyseurs, ...) fonctionnent correctement. Cette idée a déjà été fructueuse, par exemple dans le cadre du test de compilateurs C (cf. [YCER11]).

Approches alternatives

D'autres travaux adoptent un point de vue combinatoire sur l'étude de la concurrence.

En combinatoire analytique, des travaux abordent la problématique du *shuffle* de langages réguliers généralement sur des alphabets disjoints, donc sans synchronisation (cf. [GDG$^+$08] et [DPRS12]). Plus récemment, Nicolas Basset et ses coauteurs introduisent plusieurs algorithmes de génération aléatoire uniforme de mots reconnus par des automates synchronisés (cf. [BMS17]).

Une autre approche est celle proposée par Jean Mairesse et ses coauteurs sur le monoïde de traces (cf. [KMM02], [AM15]). Il s'agit d'un modèle de "vraie concurrence", dans le sens où les actions atomiques de processus en parallèle peuvent s'exécuter en même temps, sans entrelacement. Les résultats obtenus sur ce modèle vont de l'analyse de plusieurs paramètres combinatoires jusqu'à la génération aléatoire d'éléments du monoïde : analogue des traces d'exécutions.

Enfin, les travaux de Johan Oudinnet et ses coauteurs se sont concentrés sur le modèle de système de transitions tels que ceux analysés dans le cadre du *model checking* (cf. [ODG$^+$11]). Une des motivations principales est d'ailleurs de fournir un algorithme de génération aléatoire uniforme de chemins dans ces systèmes et ainsi fournir une brique efficace et non biaisée dans le cadre du *model checking* statistique introduit dans [GS05].

Ces travaux sont d'un point de vue technique assez éloignés de ceux présentés dans cette thèse. Nous avons choisi de détailler les travaux connexes plus proches au début de chaque chapitre.

Contributions de la thèse

Nous introduisons la famille de DAGs que nous nommons les diamants. Pour différentes classes de diamants nous obtenons des formules closes pour leurs séries génératrices notamment en terme de fonctions elliptiques et de fonctions dites spéciales. Pour ces séries nous obtenons des équivalents asymptotiques de leurs termes généraux. Nous présentons également une analyse en moyenne du degré de leur racine. Enfin, nous exhibons des bijections avec d'autres familles d'objets croissantes.

Sur la classe des graphes Fork-Join correspondant aux ordres Série-Parallèle, nous introduisons un opérateur dit de produit ordonné ainsi qu'un produit coloré pour spécifier les étiquetages croissants de ces graphes. Ces opérateurs désormais intégrés à la méthode symbolique ont un intérêt au-delà de la synchronisation des processus concurrents. Nous montrons notamment leur utilisation dans la spécification des arbres binaires de recherche, les permutations intervalles généralisées, les partitions d'entiers et les nombres de Bell. Sur ces opérateurs nous sommes capables de proposer des propriétés intéressantes notamment des théorèmes de transfert assez généraux. Sur les graphes Fork-Join croissants nous obtenons un équivalent asymptotique pour le nombre de graphes d'une taille donnée.

Les contributions sur les ordres partiels sans cycles sont plus mesurées. Tout d'abord nous proposons une décomposition simple des DAGs permettant de les caractériser. Cette décomposition nous permet de construire une formule symbolique (de taille linéaire) de comptage du nombre d'extensions linéaires. Les problèmes de comptage effectif et de génération aléatoire efficace restent ouverts.

Concernant la génération aléatoire uniforme nos contributions sont les suivantes. Nous proposons différents générateurs de Boltzmann pour les structures croissantes dont la série génératrice vérifie une équation différentielle d'ordre 2. Nous l'appliquons notamment aux diamants croissants. Enfin, nous proposons aussi un générateur aléatoire uniforme d'extensions linéaires pour les ordres partiels Série-Parallèle. Cet algorithme est optimal en nombre de bits aléatoires consommés.

Plan du manuscrit

Le chapitre 1 présente dans le détail l'ensemble des résultats obtenus sur les diamants croissants.

Les ordres Série-Parallèle ainsi que la classe des graphes Fork-Join sont présentés dans le chapitre 2. Les résultats sur les produits ordonné et coloré qui ne sont pas spécifiques à ces classes sont regroupés en fin de chapitre.

Le chapitre 3 est dévolu à la génération aléatoire. On rappelle ce qu'est la méthode de Boltzmann et comment on construit des générateurs de Boltzmann pour les structures croissantes. S'en suit la présentation de l'algorithme de génération aléatoire d'extensions linéaires pour les ordres Série-Parallèle. On trouvera aussi une étude pratique poussée de ces algorithmes à travers diverses expérimentations.

Ensuite dans le chapitre 4 nous présentons la décomposition et le comptage symbolique d'extensions linéaires pour les ordres partiels sans cycle. Une petite expérimentation pratique est aussi proposée.

Le manuscrit se termine par une conclusion suivi d'un panorama des perspectives de recherche.

Deux annexes sont ajoutées à la fin du manuscrit. L'annexe A rappelle un certain nombre de définitions sur les ordres partiels. L'annexe B rappelle les définitions et quelques propriétés des transformées de Laplace et Borel combinatoires.

CHAPITRE 1

DIAMANTS CROISSANTS

Les diamants croissants sont le point de départ de notre étude. Du point de vue de la combinatoire analytique, on peut les spécifier en utilisant le produit boîte [Gre83] de la méthode symbolique et l'étude de leur séries génératrices suit une méthodologie assez classique. La principale difficulté rencontrée dans l'analyse de ces différentes classes d'objets provient de l'originalité de leur séries génératrices qui s'expriment en terme de fonctions elliptiques ou encore de fonction d'erreur.

Du point de vue des programmes concurrents, ils représentent des modèles très simplifiés de programmes Série-Parallèle. La classe des diamants croissants peut être vu comme la classe non-triviale la plus simple utilisant les constructions de parallélisme et de synchronisation de processus concurrents.

1.1 Présentation

1.1.1 Introduction

Les diamants sont des graphes dirigés et acycliques possédant une unique source (un nœud sans prédécesseur) et un unique puits (un nœud sans successeur). [1]

En combinatoire analytique [FS09], nous caractérisons les objets étudiés via une décomposition (récursive) que nous synthétisons dans une spécification des objets. La classe combinatoire non étiquetée $\mathcal{D}$ d'une famille de diamant est spécifiée par

$$\mathcal{D} = \mathcal{Z} + \mathcal{Z} \times G(\mathcal{D}) \times \mathcal{Z},$$

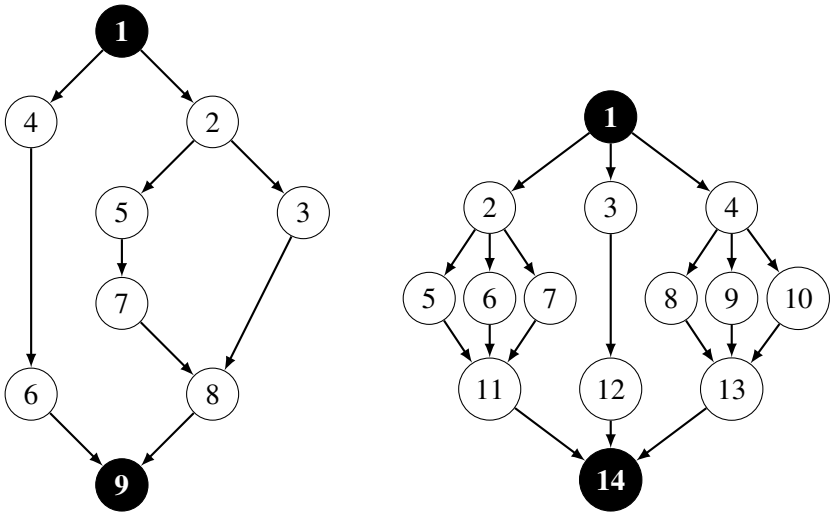
où G est la fonction des degrés et des règles de construction de la famille de diamants (binaire, ternaire, plans, cycliques, etc).

1. Les diamants que nous présentons n'ont pas de rapport avec d'autres graphes diamants que l'on peut rencontrer dans la littérature.

Dans ce cas non étiqueté, la spécification des diamants peut être interprétée comme une spécification d'arbres simplement générés (cf. [FS09]). De fait, la littérature à propos de ces arbres est vaste et le comportement de la plupart des paramètres structurels sont bien connus.

Notre étude s'intéressant à l'aspect quantitatif des exécutions de programmes concurrents, on cherche donc à étudier des classes de diamants étiquetées de manière croissante, car une exécution est en correspondance directe avec un étiquetage croissant de la structure.

Les diamants croissants sont des diamants étiquetés, c'est-à-dire que tous les nœuds ont une étiquette unique représentée par un entier naturel et tel que l'ensemble des étiquettes d'un diamant est un intervalle de $\mathbb{N}$ commençant par 1. De plus, ils sont croissants : la suite d'étiquettes obtenues le long de tout chemin de la source vers le puits est croissante et c'est les entiers de cette suite sont consécutifs.



(a) Diamant croissant binaire de taille 9

(b) Diamant croissant ternaire de taille 14

FIGURE 1.1 – Exemples de diamants

À l'aide de la méthode symbolique, on peut définir des classes de diamants croissants ainsi

$$\mathcal{F} = \mathcal{Z} + \mathcal{Z}^{\square} \star \mathcal{G}(\mathcal{F}) \star \mathcal{Z}^{\blacksquare} \tag{1.1.1}$$

où $\mathcal{G}$ est un opérateur spécifiant le type de diamants par degré et par règle de construction (planarité). Les symboles $\square$ et $\blacksquare$ sont les opérateurs boîtes (cf. [Gre83] et [FS09, p. 139]). Ils représentent, respectivement, la plus petite et la plus grande étiquette d'une structure. La Figure 1.1 présente des exemples de telles structures.

La Spécification (1.1.1) se traduit en équation fonctionnelle satisfaite par la série génératrice $f(z) = \sum_{n \geqslant 0} a_n \frac{z^n}{n!}$ de la famille de diamants :

$$f''(z) = G(f(z)), \quad \text{avec } f(0) = 0 \text{ et } f'(0) = 1. \tag{1.1.2}$$

On autorisera aussi d'autres conditions initiales, par exemple $f(0) = f'(0) = 0$.

Remarque 1:
Dans les cas que nous étudions, issus de problèmes combinatoires, la solution est garantie comme existante et unique grâce au théorème de Cauchy-Lipschitz.

Par la suite nous étudierons, dans le détail, différentes familles de diamants en fonction de l'opérateur $\mathcal{G}$: des familles polynomiales (degrés des nœuds fixés, par exemple binaire ou ternaire) et des diamants généraux, plans et non-plans (degrés des nœuds non-fixés). Les résultats que nous obtenons sur ces différentes classes sont :

— l'énumération asymptotique des objets d'une taille fixée
— le degré moyen de la source des diamants croissants de taille fixée
— et dans certains cas, des bijections avec d'autres objets combinatoires déjà connus.

1.1.2 Motivations

Du point de vue de la concurrence, les diamants sont une classe simple mais non-triviale de programmes concurrents où les constructions de parallélisme et de synchronisation sont présentes. Ils sont néanmoins peu expressifs car le mécanisme de synchronisation suit une discipline de pile : toute création de k processus en parallèles est suivi par une synchronisation de ces k mêmes processus sans poursuite possible du flot de contrôle. Néanmoins, ils ont l'intérêt d'être analysable à l'aide des outils de combinatoire analytique existants. C'est pour cela qu'ils sont un bon point de départ pour l'étude quantitative de la synchronisation.

Les bijections exhibées entre diamants croissants et objets croissants de la littérature mettent en liens nos travaux avec des travaux déjà existants. Elles permettent notamment de réutiliser les résultats obtenus pour certains paramètres sur ces objets de la littérature (et vice-versa).

1.1.3 Travaux proches

Ce travail sur les diamants a pour point de départ les travaux effectués par mes encadrants et O. Bodini, à propos de la combinatoire des programmes concurrents utilisant seulement la construction parallèle. Dans [BGP16], ils explorent la combinatoire des programmes purement parallèles où l'opérateur de composition parallèle n'est pas limité en nombre d'opérandes (vu comme une Seq ou un Set), tandis que dans (avec N. Rolin) [BGPR15], ils étudient la restriction binaire de cet opérateur. Dans tous ces cas, ils calculent le nombre moyen d'exécutions possibles de programmes d'une taille fixée, c'est-à-dire le nombre moyen d'étiquetages croissants d'arbres de taille fixée.

Concernant l'analyse de nos diamants croissants, elle est très similaire à celles des arbres croissants, étudiés par F. Bergeron, P. Flajolet et B. Salvy dans [BFS92]. Ces derniers se placent dans le cadre assez générique de l'étude de familles croissantes d'arbres simplement générés, de spécification

$$\mathcal{T} = \mathcal{Z} + \mathcal{Z}^{\square} \star \mathcal{G}(\mathcal{T}).$$

Leur analyse se concentre donc sur l'équation différentielle du premier ordre

$$f'(z) = G(f(z)), \quad \text{avec } f(0) = 0.$$

Ils utilisent alors les outils de combinatoire analytique pour calculer les asymptotiques et lois limites de différentes grandeurs : nombre moyen de feuilles, degré moyen de la racine et longueur de cheminement moyenne.

Plus récemment, Les équations différentielles rencontrées lors de notre étude des diamants croissants on aussi été rencontrées par A. Panholzer et M. Kuba lors de leurs études de modèles d'arbres doublement étiquetés croissants. Notamment, ils dérivent plusieurs formules closes, dites "des équerres", calculant le terme général de plusieurs des séries génératrices (cf. [KP16]).

1.1.4 Plan du chapitre

Nous commencerons par présenter dans le détail les différentes spécifications des objets et leur séries génératrices quand une formule close est connue. Nous ferons alors un petit détour dans notre étude quantitative en présentant des bijections avec d'autres objets connus de la littérature. Dans la section suivante, nous présenterons une étude détaillée des différents comportements asymptotiques des familles déjà présentées. Enfin, nous étudierons le degré à la racine moyen de ces familles.

1.2 Combinatoire des diamants croissants

1.2.1 Cas général

En multipliant les deux côtés de l'équation (1.1.2) par $2f'$, on obtient $2f''f' = 2G(f)f'$, ce qui implique :

$$f'(z)^2 = f'(0)^2 + \int_0^z 2G(f(t))f'(t)\mathrm{d}t = f'(0)^2 + 2\mathscr{G}(f(z)),$$

où $\mathscr{G}(z) := \int_0^z G(t)\mathrm{d}t$. Donc $f'(z) = \pm\sqrt{f'(0)^2 + 2\mathscr{G}(f(z))}$, soit :

$$\pm \int_0^{f(z)} \frac{1}{\sqrt{f'(0)^2 + 2\mathscr{G}(t)}}\,\mathrm{d}t = z. \tag{1.2.1}$$

Lemme 1:
La solution de l'équation différentielle $f'' = G(f)$ avec $f(0) = 0$ est implicitement définie par :

$$\int_0^{f(z)} \frac{1}{\sqrt{f'(0)^2 + 2\mathscr{G}(t)}}\,\mathrm{d}t = z. \tag{1.2.2}$$

Démonstration. En faisant un développement de Taylor du membre gauche de l'équation (1.2.1), on trouve que la solution négative n'est pas possible et on en conclut l'équation (1.2.2). □

La solution (1.2.2), malgré le fait qu'elle soit implicite, reste utile dans notre analyse asymptotique, même quand aucune simplification n'est possible.

Comme dans [BFS92], il faut faire attention au caractère périodique où non des séries que nous considérons.

Définition 1 (de [BFS92])**:**
Une fonction ϕ est dite périodique si il existe une fonction ψ telle que $\phi(z) = \psi(z^p)$ pour $p \geqslant 2$. Le plus grand p possible est appelé période de ϕ. Si $p = 1$, on dit de ψ qu'elle est apériodique.

Dans le cas de [BFS92], la périodicité vient de la fonction ϕ telle que $f' = \phi(f)$. Ici, on considère des équations différentielles du deuxième ordre avec des conditions initiales plus générales.

Proposition 1:
Soit une équation différentielle de diamants (1.1.2) telle que :

$$\begin{aligned} f'(z) &= \sqrt{(f'(0))^2 + 2 \cdot \mathscr{G}(f(z))} \\ &= g(f(z)). \end{aligned}$$

Si l'on note p la période de g, f est périodique de période $\mathrm{val}(f) \cdot p$ où val est la valuation [2] de f.

Démonstration. Soit p la période de g. On a alors $f'(z) = g(f(z))$. Donc, il existe ψ telle que :

$$\begin{aligned} f'(z) &= \psi(f^p(z)) \\ &= \sum_{k \geqslant 0} g_k f^{p \cdot k}(z) \\ &= \sum_{k \geqslant 0} g_k \left(\sum_{n \geqslant 0} f_n z^n \right)^{p \cdot k} \end{aligned}$$

En notant v la valuation de f, on obtient :

$$f'(z) = \sum_{k \geqslant 0} g_k \, (z^{v \cdot p})^k \left(\sum_{n \geqslant v} f_n z^{n-v} \right)^{p \cdot k}$$

On en déduit que f' est périodique de période $v \cdot p$. Une intégration symbolique du développement en série de f' nous permet de conclure que f a la même période. □

Ainsi, on adapte le lemme fondamental de [BFS92] à notre étude des diamants.

Lemme 2:
La singularité dominante ρ (réelle et positive) de la fonction f, solution de (1.1.2), est :

$$\rho = \int_0^{\rho_{\mathscr{G}}} \frac{\mathrm{d}t}{\sqrt{f'(0)^2 + 2 \int_0^t G(v) \mathrm{d}v}},$$

où $\rho_{\mathscr{G}}$ est le rayon de convergence de $\int_0^z G(t) \mathrm{d}t$.

De plus, si l'équation de la série f est apériodique, alors ρ est la seule singularité dominante de f. Si l'équation de f a une période $p \geqslant 2$, alors il existe une série apériodique $\hat{f}$ telle que $f(z) = \hat{f}(z^p)$ et $\rho^{\frac{1}{p}}$ est son unique singularité dominante.

2. La valuation d'une série est le rang de son premier terme non nul.

La preuve du lemme précédent est similaire à celle du Lemme 1 de [BFS92] avec une correction sur les bornes de l'intégrale expliquée par la proposition suivante.

Proposition 2:
Soit f solution de l'équation différentielle $f''(z) = G(f(z))$. La singularité dominante de $\int_0^z G(t)\mathrm{d}t$ est la limite de $f(z)$ quand z tend vers ρ.

Démonstration. Soit ρ la plus petite singularité réelle positive de f, dont l'existence est garantie par le théorème de Pringsheim. Par la suite on notera $f(\rho) = \lim_{z\to\rho^-} f(z)$ la limite inférieure de f en ρ (sur la droite des réelles).

On sait que G est bien définie sur $[0, f(\rho)[$. De plus, on sait que f est croissante sur $[0, \rho[$ et donc on peut définir son inverse compositionnel f^{-1} sur ce segment. On a alors $G(z) = f''(f^{-1}(z))$ pour z dans $[0, f(\rho)[$.

Montrons que $f(\rho)$ est la singularité dominante réelle de G. Par définition, G n'a pas de singularité sur $[0, f(\rho)[$. Or, $f(\rho)$ est une singularité de $f''(f^{-1}(z))$, car ρ est singularité de f et donc de f''. On en déduit donc que $f(\rho)$ est singularité de G, de plus, c'est sa plus petite singularité réelle positive. □

1.2.2 Cas avec solution explicite

Dans cette sous-section, on s'intéressera aux différentes familles de diamants pour lesquelles l'équation (1.1.2) peut être résolue explicitement. Ainsi, dans tous les cas qui suivent on trouve une formule close pour le nombre de diamants croissants d'une taille fixée.

Diamants tricolores

Les diamants croissants tricolores sont donnés par la spécification suivante

$$\mathcal{T} = \mathcal{Z} + \mathcal{Z}^{\square} \star \textsc{Seq}(\mathcal{T}) \star \textsc{Seq}(\mathcal{T}) \star \textsc{Seq}(\mathcal{T}) \star \mathcal{T}^{\blacksquare}. \tag{1.2.3}$$

Le produit de trois séquences peut être interprété comme un produit de trois séquences de trois couleurs différentes, d'où leur nom de diamants tricolores.

Théorème 1:
Les diamants croissants tricolores ont pour fonction génératrice exponentielle $T(z) = 1 - \sqrt{1-2z}$. Son rayon de convergence est $1/2$ et le nombre t_n de diamants croissants tricolores de taille n est

$$t_n = n![z^n]T(z) = \prod_{k=1}^{n}(2k-1) = (2n-1)!!$$

où le symbole !! représente la double factorielle.

Démonstration. Dans ce cas, l'équation différentielle (1.1.2) est résolue de façon élémentaire :

$$\begin{aligned}
T''(z) &= (1-T)^{-3} & \\
2\,T'(z)\,T''(z) &= 2\,T'(z)\,(1-T)^{-3} & \text{en multipliant par } 2T'(z) \\
(T'(z))^2 - 1 &= (1-T)^{-2} - 1 & \text{par intégration} \\
T'(z) - T'(z)\,T(z) - 1 &= 0 & \\
T^2(z) - 2T(z) + 2z &= 0 & \text{par intégration} \\
T(z) &= 1 - \sqrt{1-2z} & \text{après choix de la "bonne" branche}
\end{aligned}$$

On retrouve alors une fonction algébrique classique, dont on extrait facilement le terme général :

$$n![z^n]T(z) = \prod_{k=1}^{n}(2k-1) \qquad \square$$

Diamants généraux non-plans

Cette classe $\mathcal{P}$ de diamants est définie par la spécification

$$\mathcal{P} = \mathcal{Z} + \mathcal{Z}^{\square} \star \textsc{Set}(\mathcal{P}) \star \mathcal{Z}^{\blacksquare}. \tag{1.2.4}$$

Elle correspond donc au cas $\mathcal{G} = \textsc{Set}$ dans l'équation (1.1.1).

Théorème 2:
Les diamants croissants généraux non-plans ont pour fonction génératrice, la solution de (1.1.2) avec $G = \exp$. Son rayon de convergence est $\frac{\pi}{2}$ et le nombre p_n de diamants croissants généraux non-plans de taille n est

$$p_n = \frac{2^{n+1}\,(n-1)!}{\pi^n} \sum_{j=-\infty}^{+\infty} \frac{1}{(1+4j)^n}.$$

Note: On peut réécrire p_n pour $n \geqslant 2$ en terme de fonction zeta d'Hurwitz [3] $\xi(n,\,s)$:

$$p_n = (n-1)! \cdot \frac{\xi(n,\,\frac{1}{4}) + (-1)^n \xi(n,\,\frac{3}{4})}{2^{n-1}\,\pi^n}.$$

3. $\xi(n,\,s) = \sum_{k=0}^{\infty}(k+s)^{-n}$

Démonstration. En résolvant l'équation (1.1.2) avec $G = \exp$, on trouve une expression pour P' :

$$
\begin{aligned}
&P''(z) = e^{P(z)} \\
&2\,P'(z)\,P''(z) = 2\,P'(z)\,e^{f(z)} \qquad \text{en multipliant les deux côtés par } 2\,P'(z) \\
&(P'(z))^2 - 1 = 2\,e^{P(z)} - 2 \qquad \text{en intégrant entre } 0 \text{ et } z \text{ des deux côtés,} \\
&\frac{2P''(z)}{1 + (P'(z))^2} = 1 \qquad \text{car } P'' = e^P, \\
&z = 2\int_0^z \frac{P''(t)}{1 + (P'(t))^2}\mathrm{d}t \\
&z = 2\int_1^{P'(z)} \frac{1}{1+t^2}\mathrm{d}t = [\arctan(t)]_{t=1}^{t=P'(z)} \\
&P'(z) = \tan(z) + \frac{1}{\cos(z)} \qquad \text{d'où, après simplification} \\
&P(z) = -\log(1 - \sin(z))
\end{aligned}
$$

On observe que les singularités de P' sont les $\rho_k = \frac{\pi}{2} + 2k\pi$ pour $k \in \mathbb{Z}$ et notamment que sa singularité dominante est $\frac{\pi}{2}$. On utilise alors la formule de Cauchy

$$
[z^n]P'(z) = \frac{1}{2i\pi}\oint_\gamma \frac{P'(\xi)}{\xi^{n+1}}\mathrm{d}\xi
$$

valable pour un cercle γ de centre 0 de rayon plus petit que $\frac{\pi}{2}$.

On utilise alors le théorème des résidus pour calculer cette intégrale :

$$
\begin{aligned}
\frac{1}{2i\pi}\oint_{\gamma_N} \frac{P'(\xi)}{\xi^{n+1}}\mathrm{d}\xi &= Res\left[\frac{P'(z)}{z^{n+1}}; 0\right] + \sum_{j=-N}^{N} Res\left[\frac{P'(z)}{z^{n+1}}; \frac{\pi}{2} + 2j\pi\right] \\
&= [z^n]P'(z) + \sum_{j=-N}^{N} Res\left[\frac{P'(z)}{z^{n+1}}; \frac{\pi}{2} + 2j\pi\right] \qquad (1.2.5)
\end{aligned}
$$

avec γ_N un cercle de centre 0 et de rayon N

On calcule le résidu de P' en ρ_j :

$$\begin{aligned}
Res\left[\frac{P'(z)}{z^{n+1}};\rho_j\right] &= \frac{1}{z^{n+1}}(z-\rho_j)P'(z)\Big|_{z=\rho_j} \\
&\underset{z\to\rho_j}{=} \frac{z-\rho_j}{\rho_j^{n+1}}\cdot\frac{sin(\rho_j)+1}{\cos(z)} \\
&\underset{z\to\rho_j}{=} \frac{z-\rho_j}{\rho_j^{n+1}}\cdot\frac{2}{\cos(z-\rho_j+\rho_j)} \\
&\underset{z\to\rho_j}{=} \frac{z-\rho_j}{\rho_j^{n+1}}\cdot\frac{2}{-sin(z-\rho_j)} \qquad \text{par identité trigonométrique} \\
&\underset{z\to\rho_j}{=} \frac{z-\rho_j}{\rho_j^{n+1}}\cdot\frac{2}{-\left[(z-\rho_j)-\frac{(z-\rho_j)^3}{3!}+o\left((z-\rho_j)^3\right)\right]} \\
&= -\frac{2}{\rho_j^{n+1}}
\end{aligned}$$

Soit, en remplaçant dans l'équation (1.2.5) :

$$\begin{aligned}
\frac{1}{2i\pi}\oint_{\gamma_N}\frac{P'(\xi)}{\xi^{n+1}}\mathrm{d}\xi &= [z^n]P'(z)+\sum_{j=-N}^{N} Res\left[\frac{P'(z)}{z^{n+1}};\frac{\pi}{2}+2j\pi\right] \\
&= [z^n]P'(z)-\frac{2^{n+2}}{\pi^{n+1}}\sum_{j=-N}^{N}\frac{1}{(1+4j)^{n+1}}. \qquad (1.2.6)
\end{aligned}$$

On remarque que l'intégrale $\frac{1}{2i\pi}\oint_{\gamma_N}\frac{P'(\xi)}{\xi^{n+1}}\mathrm{d}\xi$ tend vers 0 quand N tend vers l'infini, ainsi :

$$\begin{aligned}
\oint_{\gamma}\frac{P'(\xi)}{\xi^{n+1}}\mathrm{d}\xi &\leqslant \oint_{\gamma}\left|\frac{P'(\xi)}{\xi^{n+1}}\right|\mathrm{d}\xi \\
&\leqslant \sup_{t\in[0,1]}\left|\frac{P'(\gamma(t))}{\gamma(t)^{n+1}}\right|\cdot\int_0^1\left|\frac{\mathrm{d}\gamma}{\mathrm{d}t}\right|\mathrm{d}t \\
&\leqslant \frac{1}{R^{n+1}}\cdot\sup_{\theta\in[0,2\pi]}\left|\tan\left(Re^{i\theta}\right)+\frac{1}{\cos\left(Re^{i\theta}\right)}\right|\cdot 2\pi R \\
&\leqslant \frac{2\pi}{R^n}\left(\tan(R)+\frac{1}{\cos(R)}\right)\underset{N\to\infty}{\to} 0
\end{aligned}$$

avec $R=\left(\frac{\pi}{2}+2N\pi+\epsilon\right)$ le point de γ_N le plus proche de la singularité ρ_N.

D'où en passant la limite (de N vers l'infini) dans l'équation (1.2.6) :

$$[z^n]P'(z)=\frac{2^{n+2}}{\pi^{n+1}}\sum_{j=-\infty}^{+\infty}\frac{1}{(1+4j)^{n+1}}.$$

Soit, après renormalisation :

$$\begin{aligned} p_n &= n!\,[z^n]P(z) = (n-1)!\,[z^{n-1}]P'(z) \\ &= \frac{2^{n+1}(n-1)!}{\pi^n} \sum_{j=-\infty}^{+\infty} \frac{1}{(1+4j)^{n+1}}. \end{aligned}$$

□

Diamants de Weierstrass

On s'intéresse maintenant aux des diamants binaires :

$$\mathcal{B} = \mathcal{Z} + \mathcal{Z}^{\square} \star (\mathcal{E} + \mathcal{B} \star \mathcal{B}) \star \mathcal{Z}^{\blacksquare}. \tag{1.2.7}$$

et plus généralement aux diamants pour lesquelles l'opérateur $\mathcal{G}$ est un polynôme de degré 2.

Premièrement, considérons le cas plus général où G est un polynôme de degré 2 :

$$f''(z) = af^2(z) + bf(z) + c. \tag{1.2.8}$$

En multipliant les deux côtés par $2f'(z)$ et intégrant, on obtient :

$$(f'(z))^2 = \tilde{a}f^3(z) + \tilde{b}f^2(z) + \tilde{c}f(z) + d.$$

pour $\tilde{a}$, $\tilde{b}$ et $\tilde{c}$ judicieusement choisis et d dépendant des conditions initiales de l'équation différentielle. On pose alors $g(z) = u \cdot f(z) + v$, en choisissant u et v judicieusement en fonction de a, b et c, et de manière à obtenir une équation de la forme :

$$(g'(z))^2 = 4g^3(z) - g_2 g(z) - g_3. \tag{1.2.9}$$

où g_2 et g_3 sont des constantes.

Or, d'après [Law13], les solutions de cette équation différentielle sont des fonctions elliptiques $\wp$ de Weierstrass. La famille de fonctions $\wp$ est l'ensemble de fonctions complexes doublement périodiques sur un réseau du plan complexe, telles que chaque cellule du réseau contienne un unique pôle d'ordre 2 en un coin. Plus formellement :

$$\wp(z;\omega_1,\omega_2) = \frac{1}{z^2} + \sum_{n^2+m^2\neq 0} \frac{1}{(z+m\omega_1+n\omega_2)^2} - \frac{1}{(m\omega_1+n\omega_2)^2}.$$

où ω_1 et ω_2 sont les deux périodes de la fonction.

La forme générale des solutions de l'équation (1.2.8) est donc :

$$f(z) = A\wp(z+C;g_2,g_3).$$

Les calculs permettant de trouver ω_1 et ω_2 à partir de g_2 et g_3, dans le cas général, sont présentés dans [AS64]. Ici, nous présentons le cas particulier des diamants binaires avec :

$$B''(z) = 1 + B(z)^2.$$

Premièrement, en utilisant le Lemme 2, on peut calculer la singularité dominante $\rho_{\mathcal{B}}$ de $B(z)$ (qui est apériodique) :

$$\rho_{\mathcal{B}} = \int_0^\infty \frac{\mathrm{d}t}{\sqrt{\frac{2}{3}t^3 + 2t + 1}}.$$

Puis, par identification, on trouve :

$$B(z) = 6\wp(z + C; -\frac{1}{3}, -\frac{1}{36}).$$

Par définition, la fonction $\wp$ a un pôle d'ordre 2 en zéro, on en déduit :

$$B(z) = 6\wp(z - \rho_{\mathcal{B}}; -\frac{1}{3}, -\frac{1}{36}).$$

On note ω_3 la somme des deux périodes fondamentales de B : $\omega_3 = \omega_1 + \omega_2$. De [AS64], on sait que les racines e_1, e_2 et e_3 du polynôme $P = 4X^3 - g_2X - g3 = 4X^3 + \frac{1}{3}X + \frac{1}{36}$ vérifient $\wp(\omega_i/2) = e_i$. Après calcul on obtient :

$$\begin{cases} e_1 = \dfrac{\sqrt[3]{4} - 1}{12\sqrt[3]{2}} - i\sqrt{3}\dfrac{1 + \sqrt[3]{4}}{12\sqrt[3]{2}} \\ e_2 = \dfrac{\sqrt[3]{4} - 1}{12\sqrt[3]{2}} + i\sqrt{3}\dfrac{1 + \sqrt[3]{4}}{12\sqrt[3]{2}} \\ e_3 = \dfrac{1}{6}\left(\dfrac{1}{\sqrt[3]{1}} - \sqrt[3]{2}\right) \end{cases}$$

De [Dav62], on sait que :

$$\omega_1 = \frac{2K(k)}{\sqrt{e_1 - e_2}} \qquad \text{et} \qquad \omega_2 = \frac{2iK'(k)}{\sqrt{e_1 - e_2}}$$

avec $k = \sqrt{\dfrac{e_3 - e_2}{e_1 - e_2}}$, et K désignant l'intégrale elliptique complète de première espèce :

$$K(k) = \int_0^1 \frac{\mathrm{d}x}{\sqrt{(1 - x^2)(1 - k^2x^2)}} \qquad \text{et} \qquad K'(k) = K(k')$$

où k' est tel que $k^2 + k'^2 = 1$.

On obtient donc un développement en série pour B et son terme général :

$$B(z) = 6\left[\frac{1}{(z - \rho_{\mathcal{B}})^2} + \sum_{(k,\ell)\in\mathbb{Z}^2\setminus\{(0,0)\}} \left(\frac{1}{(z - \rho - k\omega_1 - \ell\omega_2)^2} - \frac{1}{(k\omega_1 + \ell\omega_2)^2}\right)\right]$$

dont on extrait le terme général

$$[z^n]B(z) = 6\frac{n+1}{\rho_{\mathcal{B}}^{n+2}} \sum_{(k,\ell)\in\mathbb{Z}^2} \frac{1}{\left(1 + \frac{k\omega_1}{\rho_{\mathcal{B}}} + \frac{\ell\omega_2}{\rho_{\mathcal{B}}}\right)^{n+2}} \tag{1.2.10}$$

Diamants de Jacobi

Nous considérons maintenant le cas des diamants ternaires, sans diamants de taille 1 ($T'(0) = 0$) :

$$\mathcal{T} = \mathcal{Z}^{\square} \star (\mathcal{E} + \mathcal{T} \star \mathcal{T} \star \mathcal{T}) \star \mathcal{Z}^{\blacksquare}. \tag{1.2.11}$$

et plus généralement le cas des diamants où G est un polynôme de degré 3, que l'on supposera unitaire sans perte de généralité :

$$f''(z) = f^3(z) + af^2(z) + bf(z) + c.$$

Pour ce faire, on utilise la méthode de résolution proposée dans [Dav62].

On multiplie les deux côtés de la précédente équation par $2f'$, puis en on intègre, pour obtenir :

$$(f'(z))^2 = f^4(z) + \hat{a}f^3(z) + \hat{b}f^2(z) + \hat{c}f(z) + d$$

où d dépend des conditions initiales de l'équation.

On calcule les 4 racines α, β, γ et δ du polynôme caractéristique associé à cette équation : $X^4 + \hat{a}X^3 + \hat{b}X^2 + \hat{c}X + d$. Puis, on définit la fonction g telle que :

$$g^2 = \frac{\beta - \delta}{\alpha - \delta} \cdot \frac{f - \alpha}{f - \beta}. \tag{1.2.12}$$

Après calcul, on trouve que g est solution d'une équation différentielle de la forme :

$$g'(z) = M\sqrt{(1 - g^2(z))(1 - k^2 g^2(z))}.$$

Or, les solutions de cette équation sont de la forme de la fonction sinus de Jacobi :

$$g(z) = \mathrm{sn}(Mz; k).$$

Comme la fonction $\wp$ de Weierstrass, la fonction sinus de Jacobi est définie sur le plan complexe et est doublement périodique sur un réseau. Dans le cas de la fonction sn, il y a deux singularités de type pôles, et d'ordre 1, pour chaque cellule du réseau.

Il ne reste donc plus qu'à inverser l'équation (1.2.12) pour obtenir une expression de f en fonction de sn. L'étude de f doit alors se faire au cas par cas.

Revenons donc sur notre cas particulier des diamants ternaires :

$$\begin{cases} T(0) = T'(0) = 0 \\ T''(z) = 1 + T^3(z) \end{cases}$$

En utilisant la méthode précédemment décrite, on obtient :

$$T(z) = \frac{2^{2/3}\mathrm{sn}(Mx; k)^2}{1 - j - \mathrm{sn}(Mx; k)^2} \tag{1.2.13}$$

avec $k^2 = e^{i\pi/3}$, $M = 2^{-5/6}(1 - j)^{1/2}$ où j est la racine troisième de l'unité $e^{i\frac{2\pi}{3}}$.

Par définition, la fonction $\mathrm{sn}(\cdot;\, k)$ est doublement périodique. On note K et K' ses quarts de période, définis par :

$$K = \int_0^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}} \text{ et } K' = \int_0^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k'^2t^2)}}$$

avec k' le comodule vérifiant $k'^2 = 1 - k^2$. La fonction sn^2 étant $2K$ et $2iK'$ périodique, T est $\frac{2K}{M}$ et $\frac{2iK'}{M}$ périodique. On notera dans la suite $Q_1 = \frac{K}{M}$ et $Q_2 = \frac{iK'}{M}$.

De l'équation (1.2.13), on déduit directement que les singularités de T ne viennent pas des pôles de sn mais de l'annulation de son dénominateur. On trouve donc que la singularité dominante $\rho_{\mathcal{T}}$ de T est solution de $1 - j - \mathrm{sn}(Mz;\, k)^2 = 0$:

$$\rho_{\mathcal{T}} = \frac{1}{M}\mathrm{Arcsn}(\sqrt{1-j}) = \frac{1}{M}\int_0^{\sqrt{1-j}} \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}}.$$

En utilisant diverses identités et la périodicité de sn on trouve que les singularités de T se trouvent aux points congrus à $\rho_{\mathcal{T}}$ modulo $2Q_1$ et $2\rho_{\mathcal{T}}$ modulo $2Q_2$.
De plus, toujours grâce à [Law13], on a que les résidus de T en ces deux différents types de pôles sont respectivement $-\sqrt{2}$ et $\sqrt{2}$. Comme dans le cas des Diamants généraux non-plans, on utilise alors le théorème des résidus pour pouvoir calculer la formule de Cauchy donnant le terme général de T. Après calcul, on obtient :

$$\begin{aligned}
[z^n]T(z) &= \sum_{(k,\ell)\in\mathbb{Z}^2} \frac{\sqrt{2}}{(\rho + 2kQ_1 + 2\ell Q_2)^{n+1}} - \frac{\sqrt{2}}{(2\rho + 2kQ_1 + 2\ell Q_2)^{n+1}} \\
&= \frac{\sqrt{2}}{\rho^{n+1}} \sum_{(k,\ell)\in\mathbb{Z}^2} \frac{1}{\left(1 + \frac{3k}{2} + i\frac{\sqrt{3}}{2}(k+2\ell)\right)^{n+1}} - \frac{1}{\left(2 + \frac{3k}{2} + i\frac{\sqrt{3}}{2}(k+2\ell)\right)^{n+1}}.
\end{aligned}$$

1.3 Bijections

Plusieurs familles de diamants croissants dont certaines sont énumérées par les même séries que des objets déjà connus. Ici, nous exhibons des bijections entre ces familles. Dans chacune des bijections présentées, l'idée fondamentale est que les structures que nous considérons peuvent être décomposées en arbre. L'intérêt de présenter ces bijections est double :

— cela permet de donner une méthode plus ou moins systématique de construction de bijections avec des diamants croissants,

— cela permet d'utiliser les calculs de paramètres moyens pour d'autres structures.

Ces différentes bijections montrent l'intérêt de notre étude sur les diamants croissants en présentant d'autres objets pour lesquels nos résultats peuvent être généralisés.

1.3.1 Cactus triangulaires avec ponts

La première famille à laquelle nous nous intéressons est la famille des diamants généraux plans. Leur spécification est la suivante :

$$\mathcal{F} = \mathcal{Z} + \mathcal{Z}^{\square} \star \textsc{Seq}\,(\mathcal{F}) \star \mathcal{Z}^{\blacksquare} \tag{1.3.1}$$

et leur fonction génératrice est solution de l'équation différentielle suivante :

$$\begin{cases} F''(z) = \frac{1}{1-F(z)} \\ F(0) = 0 \quad \text{et} \quad F'(0) = 1 \end{cases} \tag{1.3.2}$$

Les premiers termes de la série (n-ième terme multiplié par $n!$) sont :

$$(1, 1, 1, 3, 13, 77, 573, 5143, 54025, 650121, 8817001, 133049339, \dots)$$

aussi connue sous le nom de `A032035` dans OEIS (*Online Encyclopedia of Integer Sequence*). Or, cette suite énumère aussi les graphes cactus triangulaires avec ponts, enracinés et croissants. Commençons donc par vérifier que les deux classes d'objets ont bien la même série génératrice.

Définition 2:
Un graphe cactus est un graphe connexe, tel que toutes paires de cycles de ce graphe a au plus un nœud en commun.

Dans notre cas, les graphes que nous considérons sont aussi :
enracinés : le nœud d'étiquette 1 est distingué des autres et appelé racine,
croissants : les graphes sont dirigés, chacun de leurs nœud a une étiquette différente de celles des autres. De plus, les étiquettes le long d'un chemin d'un graphe sont ordonnées de manière croissante
triangulaires avec ponts : les cycles sont de taille deux (des arcs) ou trois (des triangles).
Un exemple de tel graphe cactus est donné page 20. Enfin, on peut donner une spécification d'une telle classe $\mathcal{G}$ de graphes :

$$\mathcal{G} = \mathcal{Z}^{\square} \star \textsc{Set}\left(\mathcal{G} + \underset{=2}{\textsc{Cyc}}\left(\mathcal{G}\right)\right).$$

On lira cette spécification ainsi : un graphe de $\mathcal{G}$ est soit un nœud isolé (un nœud avec un ensemble vide), soit un nœud de plus petite étiquette, racine d'un ensemble de cycle de taille trois (un nœud et un cycle de taille deux) et d'un ensemble de graphes de $\mathcal{G}$. La direction des arcs de ce graphe est naturellement induite par l'étiquetage croissant de ses nœuds.

Via la méthode symbolique, on en déduit que la fonction génératrice G est solution de l'équation différentielle :

$$\begin{cases} G(0) = 0 \\ G'(z) = e^{G(z)+\frac{G(z)^2}{2}} \end{cases} \tag{1.3.3}$$

En remarquant que $F'(z) - 1$ satisfait aussi l'équation différentielle (1.3.3) on en déduit que ces deux classes d'objets sont bien comptées par les même séries génératrices, à un décalage près.

Les spécifications de ces deux types d'objets sont très différentes : les diamants sont plans (opérateur Seq) tandis que les cactus ne le sont pas (opérateur Set). Pour trouver une bijection, on va donc manipuler les équations différentielles (1.3.2) et (1.3.3) en les interprétant de manière combinatoire.

Étant donné que $G = F' - 1$, une intuition consiste à dériver les deux équations différentielles pour obtenir :

$$F''' = F' \cdot \frac{1}{(1-F)^2} \qquad \text{et} \qquad G'' = (1+G) \cdot \left(G'\right)^2$$

En interprétant combinatoirement l'opérateur de dérivation comme l'opération de suppression d'un nœud d'étiquette particulière (comme dans le cas des *espèces* combinatoires [BLL98]), on obtient deux nouvelles décompositions de nos structures :

— $F''' = F' \cdot \frac{1}{(1-F)^2}$: si l'on supprime les nœuds d'étiquettes $1, 2$ et n à un diamant croissant (F'''), on obtient un diamant croissant amputé de sa racine (F') avec une séquence de diamants croissants à sa droite ($\frac{1}{1-F}$) et une autre à sa gauche ($\frac{1}{1-F}$),

— $G'' = (1 + G) \cdot (G')^2$: un cactus 2-3 enraciné et croissant où l'on supprime les nœuds d'étiquettes 1 et 2 est scindé en trois cactus : les cactus enracinés en 1 et 2 amputés de leur racine ($G' \cdot G'$) et un cactus (G), vide (1) si $1, 2$ ne sont pas dans un cycle de taille trois.

On peut transcrire cette description sous la forme de la fonction Ψ (ci-dessous) qui transforme récursivement des cactus en diamants. Afin de lire la définition suivante, notez que (et ces notations seront reprises par la suite pour d'autres bijections) :

— a, b désignent les deux étiquettes les plus petites de la structure et $a < b$,

— c est l'étiquette du troisième noeud dans un cycle contenant les noeuds d'étiquettes a et b,

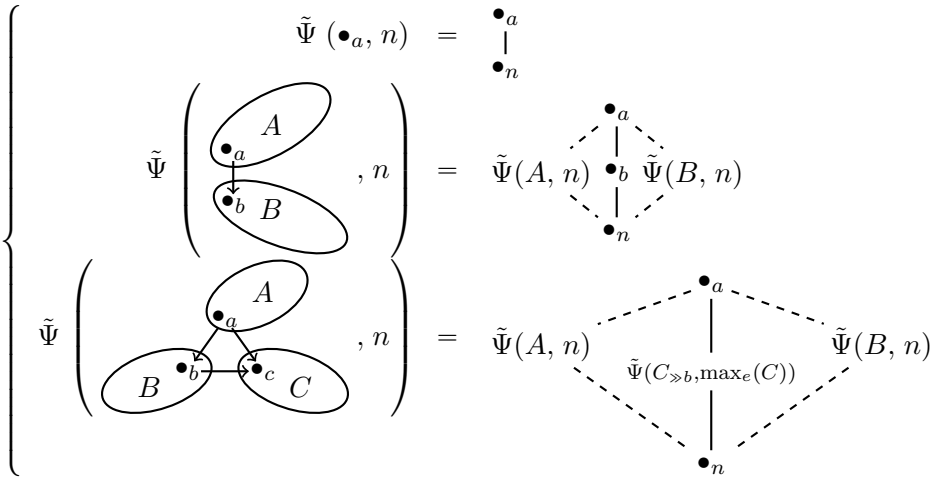
— les sous-graphes A, B et C sont disjoints,

— les lignes en tirets dénotent des contractions d'arcs entre des sommets : les deux sommets aux extrémités d'une ligne en tirets sont fusionnés en un seul dont l'étiquette est celle de la racine ou du puits fusionné,

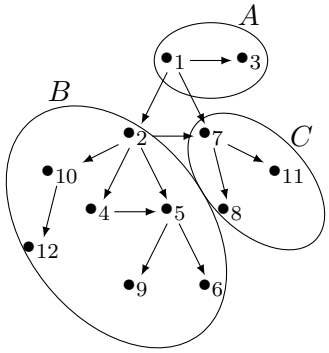
— $\max_e(T)$ désignent la plus grande étiquette du graphe T.

— $T_{\gg x}$ est le graphe T tel que ses étiquettes (dans l'ordre croissant) $(e_1, \ldots, e_n)$ sont remplacées par $(x, e_1, \ldots, e_{n-1})$

— on pose $\tilde{\Psi}(A, |A| + 1) = \Psi(A)$:



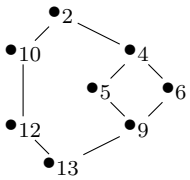
Dans le cas où $\tilde{\Psi}(A, \cdot)$ ne serait qu'une chaîne de deux nœuds, la contraction des nœuds fait disparaître l'arc restant. Le deuxième paramètre n de Ψ, permet d'ajuster l'étiquette du puits du diamant obtenu à partir de C, dans le dernier cas. Par soucis pédagogique, illustrons cette construction sur un exemple :

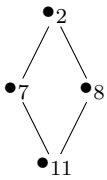
On veut calculer $\Psi(T)$ où $T =$ 

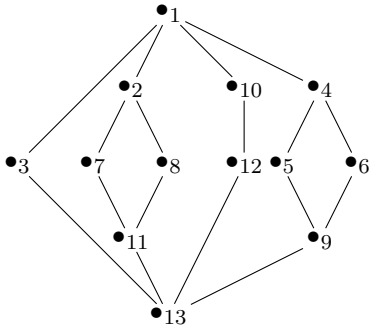
.

Il faut donc calculer $\tilde{\Psi}(A,\, 13)$, $\tilde{\Psi}(B,\, 13)$ et $\tilde{\Psi}(C_{\gg 2},\, 11)$:

$\tilde{\Psi}(A,\, 13) =$

$\tilde{\Psi}(B,\, 13) =$ 

$\tilde{\Psi}(C_{\gg 2},\, 11) =$ 

D'où, en recollant les morceaux : $\Psi(T) =$ 

En pratique, cette bijection est simple à implémenter de manière efficace : avec complexité linéaire en nombre de nœuds. Elle permet notamment, en utilisant les algorithmes développés durant ce travail de thèse, d'obtenir un algorithme de génération aléatoire uniforme efficace de graphes cactus triangulaires croissants.

1.3.2 Arbres unaire-binaires non-plans

En utilisant la même méthode, consistant à manipuler les équations différentielles pour trouver des similarités entre objets combinatoires, on obtient une bijection entre les diamants généraux, non-plans et les arbres unaire-binaires croissants plans. Dans le cas de cette bijection, les deux types de structures ne sont pas plongés dans le plan.

La spécification de la classe $\mathcal{A}$ des arbres unaires-binaires non-plans est une spécification d'arbre croissant simplement généré, comme dans les cas traités par [BFS92].

$$\mathcal{A} = \mathcal{Z}^{\square} \star \underset{\leqslant 2}{\textsc{Set}}\,(\mathcal{A})$$

Toujours grâce à la méthode symbolique, on obtient une équation différentielle dont la solution est la série génératrice exponentielle A de ces arbres :

$$\begin{cases} A(0) = 1 \\ A'(z) = 1 + A(z) + \frac{A(z)^2}{2} \end{cases}$$

Les premiers termes de la suite $(n![z^n]A(z))_{n\geqslant 0}$ sont

$$(1, 1, 1, 2, 5, 16, 61, 272, 1385, 7936, 50521, 353792, \ldots).$$

Cette suite est référencée comme la suite `A000111` dans OEIS et dénombre aussi les nombres d'Euler ou le nombre d'extensions linéaires des ordre partiels "zig-zag" (les permutations alternantes). De fait, cette suite dénombre aussi le nombre de diamants non-plans généraux, de taille n, et de spécification :

$$\mathcal{P} = \mathcal{Z} + \mathcal{Z}^{\square} \star \textsc{Set}(\mathcal{P}) \star \mathcal{Z}^{\blacksquare}$$

Comme précédemment, un petit jeu de manipulation d'équations nous donne les relations suivantes :

$$P''' = P' \cdot P'' \qquad \text{et} \qquad A'' = (1 + A) \cdot A'$$

avec $P' = 1 + A$.

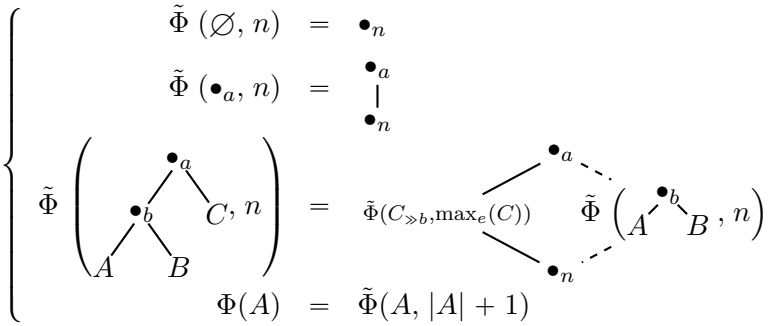
On obtient donc les décompositions suivantes :

— $P''' = P' \cdot P''$: un diamant général non-plan dont on supprime les deux nœuds de plus petites étiquettes. Celui de plus grande étiquette (P''') est scindé en un diamant général non-plan sans racine pour le diamant enraciné en le nœud d'étiquette 2 (P') et un ensemble de diamants (ou un diamant sans racine ni puits) pour le diamant enraciné en 1 (P''),

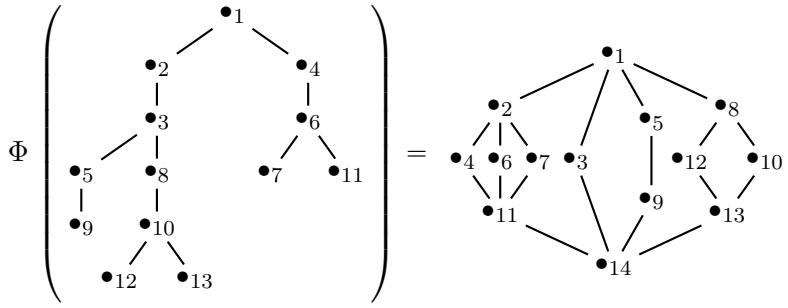
— $A'' = (1 + A) \cdot A'$: si l'on supprime les deux plus petites étiquettes d'un arbre unaire-binaire non-plan (A''), il reste un de ses sous arbres sans racine, celui qui était enraciné en 2 (A') et un arbre unaire-binaire (A), vide si la racine initiale était unaire (1).

De la même façon, que précédemment, on utilise cette décomposition pour construire une bijection Φ.

On suppose que a et b sont les plus petites étiquettes.



Comme précédemment, voici un exemple :



1.3.3 Arbres généraux plans

Dans ce dernier cas, on présente une bijection entre les diamants croissants tricolores et les arbres généraux plans et croissants.

On rappelle que la spécification de la classe combinatoire $\mathcal{T}$ des diamants croissants tricolores est la suivante :

$$\mathcal{T} = \mathcal{Z} + \mathcal{Z}^{\square} \star \textsc{Seq}(\mathcal{T}) \star \textsc{Seq}(\mathcal{T}) \star \textsc{Seq}(\mathcal{T}) \star \mathcal{T}^{\blacksquare}$$

et que la spécification de la classe combinatoire $\mathcal{C}$ des arbres généraux plans et croissants est :

$$\mathcal{C} = \mathcal{Z}^{\square} \star \textsc{Seq}(\mathcal{C}).$$

Dans ce cas, les deux séries sont les même et leurs premiers termes (normalisés par une factorielle) sont

$$(1, 1, 3, 15, 105, 945, 10395, 135135, 2027025, 34459425, 654729075, \dots)$$

connus sous le nom de `A001147` dans OEIS. On sait que le terme général de ces séries est la double factorielle des nombres impaires :

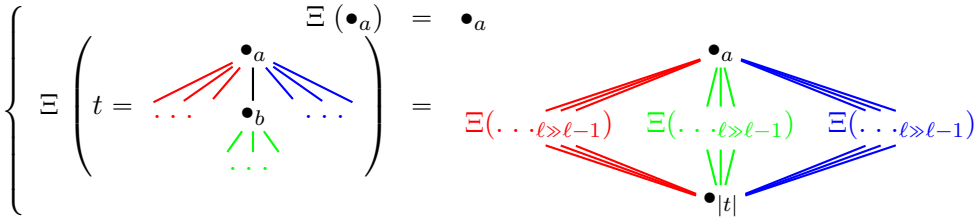
$$n!\,[z^n]C(z) = n!\,[z^n]T(z) = (2n-1)!! = \prod_{k=1}^{n}(2n-1).$$

On obtient donc directement les deux décompositions suivantes :

— $C'' = \frac{C'}{(1-C)^2}$: si l'on supprime la racine et le nœud d'étiquette 2 d'un arbre général plan et croissant, on obtient deux séquences d'arbres de Catalan croissants séparées par une troisième séquence fille du nœud d'étiquette 2

— $T'' = \frac{1}{(1-T)^3}$: un diamant croissant tricolore peut être décomposé en une racine, un puits et trois séquences de diamants tricolores de trois couleurs différentes

De cette décomposition on obtient simplement une bijection entre les deux types de structures. Mais encore, la gestion des étiquettes demande un peu de précaution : le nœud d'étiquette 2 est envoyé sur le puits du diamant et toutes les autres étiquettes sont décrémentées. Dans la description, ci-dessous, de la bijection Ξ, on utilise l'annotation $\ell \gg \ell - 1$ pour le signifier. De plus, par soucis esthétique, on fait l'amalgame entre la transformation d'une séquence et la séquence des transformations : $\Xi((A_1, \dots, A_n)) = (\Xi(A_1), \dots, \Xi(A_n))$.



Note: Nous n'avons pas présenté les preuves que les fonctions Φ, Ψ et Ξ sont des bijections car dans les trois cas les preuves sont des inductions structurelles simples.

1.4 Comportement asymptotique

1.4.1 Familles polynomiales

Dans le cas des familles de diamants ayant pour opérateur G un polynôme comme par exemple les diamants de Weierstrass ou de Jacobi, le comportement asymptotique du terme général est facilement obtenu en suivant la même méthode que [BFS92].

Diamants binaires

Dans le cas des diamants binaires que nous avons étudié G est le polynôme $1 + X^2$ et est donc apériodique au sens de [BFS92]. On peut donc utiliser le Lemme 2 directement.

Commençons par calculer la singularité dominante $\rho_{\mathcal{B}}$ de $B(z)$:

$$\rho_{\mathcal{B}} = \int_0^\infty \frac{\mathrm{d}t}{\sqrt{\frac{2}{3}t^3 + 2t + 1}}.$$

On rappelle que dans ce cas l'équation (1.2.2) nous donne :

$$z = \int_0^{B(z)} \frac{\mathrm{d}t}{\sqrt{\frac{2}{3}t^3 + 2t + 1}}.$$

En utilisant ces deux égalités, on obtient le développement asymptotique de B :

$$z - \rho_{\mathcal{B}} = \int_{B(z)}^\infty \frac{\mathrm{d}t}{\sqrt{\frac{2}{3}t^3 + 2t + 1}} \underset{z\to\rho_{\mathcal{B}}}{\sim} \int_{B(z)}^\infty \frac{\mathrm{d}t}{\sqrt{\frac{2}{3}}t^{\frac{3}{2}}} = \frac{\sqrt{6}}{\sqrt{B(z)}}$$
$$\text{et donc} \quad B(z) \underset{z\to\rho_{\mathcal{B}}}{\sim} \frac{6}{(z-\rho_{\mathcal{B}})^2}.$$

Soit en utilisant le théorème de transfert standard de [FO90] :

$$n![z^n]B(z) \underset{n\to\infty}{\sim} \frac{6(n+1)!}{\rho_{\mathcal{B}}^{n+2}}.$$

On rappelle que ce résultat peut directement obtenu à partir de la formule (1.2.10) qui est une formule close pour le terme général.

Diamants Ternaires

Dans le cas des diamants ternaires, l'asymptotique n'est pas aussi direct à calculer. La condition initiale $T'(0) = 0$ implique la 6-périodicité de T :

— $z \mapsto \sqrt{T'(0)^2 + 2\mathscr{G}(z)}$ est de période 3 :

$$\sqrt{T'(0)^2 + 2\mathscr{G}(z)} = \sqrt{2\left(z + \frac{z^4}{4}\right)}$$
$$= \sqrt{2z} \sum_{n \geqslant 0} \frac{\prod_{k=0}^{n-1}\left(\frac{1}{2} - k\right)}{8^n n!}(z^3)^n.$$

— la valuation de f est 2

$\Rightarrow$ la période de f est donc $2 \cdot (3-1) + 2 = 6$.

En utilisant la même méthode que pour les diamants binaires, on obtient :

$$z - \rho_{\mathcal{T}} = \int_{T(z)}^{\infty} \frac{2\mathrm{d}t}{\sqrt{2t^4 + 8t}} \underset{z \to \rho_{\mathcal{T}}}{\sim} \int_{T(z)}^{\infty} \frac{\mathrm{d}t}{\sqrt{2t^2}} = \frac{\sqrt{2}}{T(z)}$$

$$\text{et donc} \quad T(z) \underset{z \to \rho_{\mathcal{T}}}{\sim} \frac{\sqrt{2}}{(z - \rho_{\mathcal{T}})}.$$

Or ce raisonnement est le même pour les 6 singularités dominantes de T. En sommant les contributions de chacune et en utilisant le théorème de transfert, on obtient

$$[z^n]T(z) \underset{n \to \infty}{\sim} \frac{6\sqrt{2}}{\rho_{\mathcal{T}}^{n+1}}.$$

Cas des diamants polynomiaux

Dans le cas général d'une famille de diamants définie par la spécification (1.1.1) avec G un polynôme de degré m, on obtient le résultat théorème suivant.

Théorème 3:
Soient $G = \sum_{k=0}^{m} g_k X^k$ un polynôme et f la série génératrice de diamants polynomiaux spécifiés par :

$$\mathcal{F} = \mathcal{Z} + \mathcal{Z}^{\square} \star G(\mathcal{F}) \star \mathcal{Z}^{\blacksquare}. \tag{1.4.1}$$

que l'on suppose apériodique.

Quand n tend vers l'infini on a :

$$f_n = n! \left(\frac{\sqrt{2(m+1)}}{(m-1)\sqrt{g_m}}\right)^{\frac{2}{m-1}} \frac{n^{-\frac{m-3}{m-1}}}{\Gamma(\frac{2}{m-1})} \rho^{-n-\frac{2}{m-1}} \left(1 + \mathcal{O}\left(n^{-\frac{4}{m-1}}\right)\right),$$

pour $m \geqslant 2$, où ρ est donné par le théorème 2, c'est-à-dire :

$$\rho = \int_0^{\infty} \frac{\mathrm{d}t}{\sqrt{1 + 2\int_0^t \sum_{k=0}^{m} \frac{g_k}{k+1} v^{k+1} \mathrm{d}v}}.$$

Remarquons que l'équivalent asymptotique de f_n ne dépend pas des conditions initiales dans ce cas.

Remarque 2:
Dans le cas périodique, il faut multiplier cet équivalent asymptotique par le nombre de singularités présentes sur le cercle de convergence.

Démonstration. Comme dans les cas précédents, on part du théorème 2 pour obtenir :

$$\begin{aligned}\rho - z &= \int_{f(z)}^{\infty} \frac{1}{\sqrt{1 + 2\sum_{0\leqslant j\leqslant m} \frac{g_j}{j+1} t^{j+1}}}\,\mathrm{d}t \\ &\underset{f(z)\to\infty}{=} \frac{\sqrt{m+1}}{(m-1)\sqrt{2g_m}}\, f(z)^{-\frac{m-1}{2}} - \frac{g_{m-1}\sqrt{m+1}}{m g_m^{3/2}}\, f(z)^{-\frac{m+1}{2}} + \cdots ,\end{aligned}$$

quand z tend vers ρ. On peut alors inverser l'équation pour obtenir :

$$f(z) = \left(\frac{(m-1)\sqrt{g_m}}{\sqrt{2(m+1)}}\right)^{-\frac{2}{m-1}} (\rho - z)^{-\frac{2}{m-1}} \left(1 + \mathcal{O}\left(|\rho - z|^{\frac{2}{m-1}}\right)\right),$$

quand z tend vers ρ. On utilise alors le théorème de transfert standard pour conclure. □

1.4.2 Cas des diamants plans

Ici, on s'intéresse à la famille des diamants généraux plans : $\mathcal{G} = \textsc{Seq}$. Dans ce cas, la méthode est la même mais l'analyse est plus compliquée car elle fait apparaître des fonctions dites spéciales.

Théorème 4:
Le nombre f_n de diamants généraux plans de taille n vérifie, quand n tend vers l'infini :

$$f_n = \frac{(n-1)!\,\rho^{1-n}}{n\sqrt{2\log n}} \left(1 + \frac{\log\log n - 3 - 2\gamma + \log 2 + 2\log\rho}{8\log n} + \mathcal{O}\left(\frac{(\log\log n)^2}{(\log n)^2}\right)\right).$$

En fait, on peut prouver un développement complet de la forme :

$$[z^n]F(z) = \frac{\rho^{1-n}}{n^2\sqrt{2\log n}} \left(\sum_{0\leqslant k<K} \frac{P_k(\log\log n)}{(\log n)^k} + \mathcal{O}\left(\frac{(\log\log n)^K}{(\log n)^K}\right)\right),$$

où les P_k sont des polynômes (de degré k) pouvant être calculés symboliquement.

Démonstration. On définit $g = F' - 1$, et l'on remarque que $g' = e^{g^2/2+g}$. On peut alors réécrire l'équation (1.2.2) :

$$z = \int_0^{g(z)} e^{-\frac{t^2}{2}-t}\mathrm{d}t \;=\; \sqrt{2e}\int_{\frac{1}{\sqrt{2}}}^{\frac{g(z)+1}{\sqrt{2}}} e^{-u^2}\mathrm{d}u. \tag{1.4.2}$$

En utilisant le Lemme 2, on obtient ρ la singularité dominante de g :

$$\rho = \sqrt{2e} \int_{\frac{1}{\sqrt{2}}}^{\infty} e^{-u^2} \mathrm{d}u \tag{1.4.3}$$

De plus ρ est un pôle : l'équation différentielle vérifiée par g suffit à dire que g n'est pas bornée sur $\mathbb{R}^+$. On pose donc $\alpha \in \mathbb{C} \cup \{\infty\}$ tel que $\lim_{x \to \alpha} g(x) = +\infty$. En le substituant dans l'équation (1.4.2), on obtient

$$\sqrt{2e} \int_{\frac{1}{\sqrt{2}}}^{\infty} e^{-u^2} \mathrm{d}u \ = \ \alpha.$$

Le côté gauche est clairement convergent et égal à ρ. Donc $\rho < \infty$ et $\lim_{x \to \rho} g(x) = +\infty$ est suffisant pour conclure que ρ est un pôle.

On reconnaît la fonction d'erreur $\mathrm{erf}(x) = \int_0^x e^{-t^2} \mathrm{d}t$. Alors, en utilisant la même méthode que dans les cas précédents :

$$\rho - z = \sqrt{2e} \int_{\frac{g(z)+1}{\sqrt{2}}}^{\infty} e^{-t^2} \mathrm{d}t = \sqrt{\frac{e\pi}{2}} \left(1 - \mathrm{erf}\left(\frac{g(z)+1}{\sqrt{2}}\right)\right). \tag{1.4.4}$$

On utilise alors un développement asymptotique standard [AS64] de la fonction d'erreur pour obtenir :

$$\rho - z \underset{z \to \rho}{\sim} \sqrt{e} \frac{\exp(-\frac{F'(z)^2}{2})}{F'(z)} \sum_{n=0}^{\infty} (-1)^n \frac{(2n-1)!!}{F'(z)^{2n}}.$$

Alors, en gardant le terme dominant de la somme et en utilisant la relation $F'^2 - 1 = -2\log(1-F)$ (obtenu par multiplication par $2F'$, puis intégration de $F'' = \frac{1}{1-F}$) :

$$\log(\rho - z) \underset{z \to \rho}{\sim} \frac{1 - F'(z)^2}{2} - \log(F'(z)) \underset{z \to \rho}{\sim} \log(1 - F(z)).$$

On se sert alors de cette relation comme *ansatz* en substituant $\log(1 - F(z))$ par $\log(\rho - z)$ dans (1.4.4), en prenant de plus en plus de termes de la somme. On obtient alors la série de calcul suivant pouvant être développé aussi loin que souhaité :

$$\begin{aligned}
\log(1 - F(z)) &\underset{z \to \rho}{\sim} \log(\rho - z) - \frac{1}{2} \log\log(\frac{1}{1 - F(z)}) + \frac{1}{2} \log 2 \\
&\underset{z \to \rho}{\sim} \log(\rho - z) - \log\log \frac{1}{\rho - z} - \frac{1}{2} \log 2 \\
&\quad + \frac{1}{4} \frac{\log 2 + \log\log \frac{1}{\rho - z}}{\log(\rho - z)} + \mathcal{O}\left(\left(\frac{\log\log(\rho - z)}{\log(\rho - z)}\right)^2\right).
\end{aligned}$$

Pour conclure la preuve, on utilise le théorème de transfert standard (cf. [FS09]) pour dériver l'asymptotique de f_n. □

On remarquera que la forme de l'asymptotique est assez originale et que le terme d'erreur a une décroissance extrêmement lente. En fait, on pourrait améliorer ce développement en utilisant un développement en fonction W de Lambert à la place de la fonction erf.

1.5 Degré à la racine

On détaille ici l'analyse du degré à la racine moyen de différentes classes de diamants. Cela permet de donner un exemple d'analyse d'un paramètre typique. De plus, les résultats obtenus diffèrent de ceux obtenus pour les arbres croissants, ce qui montre bien l'originalité du comportement des diamants croissants.

1.5.1 Cas général

Soit $\mathcal{F}$ une famille de diamants croissants de spécification (1.1.1) et de série génératrice f. On s'intéresse maintenant à l'analyse asymptotique du degré moyen de la racine de tels graphes. On rappelle que le degré d'un nœud est le nombre d'arcs entrants et sortants de celui-ci.

Soit, $R(z,\, t)$ la fonction génératrice bivariée satisfaisant :

$$R(z,\, t) = \sum_{n,r\geqslant 0} a_{n,r}\; t^r \frac{z^n}{n!},$$

où $a_{n,r}$ est le nombre de diamants de taille n dont la racine a un degré r. Cette fonction satisfait l'équation suivante, obtenue en marquant les fils de la racine par une variable t :

$$R(z,\, t) = z + G(0)(t-1)\frac{z^2}{2} + \int_0^z \int_0^u G(t\cdot f(v))\mathrm{d}v\mathrm{d}u.$$

Le terme $G(0)(t-1)\frac{z^2}{2}$ permet de juste de marquer la chaîne d'atomes de taille 2, si il y en a (quand $G(0) \neq 0$).

Le degré moyen de la racine est donc obtenu en calculant la moyenne pondérée des $a_{n,r}$, c'est-à-dire en calculant $\dfrac{\sum_{r=0}^{n} r\cdot a_{n,r}}{f_n}$. Si l'on note R_n la variable aléatoire correspondant à l'arité d'un diamant croissant de taille n, on obtient donc la formule classique ([FS09, chapitre 3]) suivante qui est juste une réécriture de la précédente en terme de séries génératrices :

$$\mathbb{E}[R_n] = \frac{[z^n]\; \frac{\partial R}{\partial t}\big|_{t=1}}{[z^n]f(z)}.$$

Par la suite on notera $r(z) = \frac{\partial R}{\partial t}\big|_{t=1}$ et on remarque que r est solution de l'équation différentielle :

$$r''(z) = G(0) + f(z)\cdot G'(f(z)), \quad \text{avec } r(0) = 0 \text{ et } r'(0) = 0. \tag{1.5.1}$$

Du fait que r'' ne dépend que de f et G, on obtient simplement les équivalents asymptotiques recherchés pour calculer $\mathbb{E}R_n$, en utilisant les résultats de la section 1.4.

1.5.2 Diamants tricolores

Théorème 5:
Soit $\mathcal{T}$ la famille de diamants définies par la spécification (1.2.3), avec $G(z) = (1-z)^{-3}$. Quand n tend vers ∞, le degré moyen de la racine d'un diamants de $\mathcal{T}$ de taille n est :

$$\mathbb{E}[R_n] \underset{n\to\infty}{=} \frac{3}{2}\sqrt{\pi\, n} + O\,(1)\,.$$

Démonstration. On connaît la fonction génératrice de $\mathcal{T}$: $T(z) = 1 - \sqrt{1-2z}$. Ainsi, en utilisant l'équation (1.5.1), on obtient :

$$r(z) = \frac{z^2}{2} + \frac{3z}{2} - 3 - \frac{3}{4}\log(1-2z) + 3\sqrt{1-2z}.$$

Puis on applique le théorème de transfert standard pour obtenir le résultat attendu. □

1.5.3 Diamants généraux non-plans

Théorème 6:
Soit $\mathcal{P}$ la famille des diamants définies par la spécification (1.2.4), avec $G(z) = e^z$. Quand n tend vers ∞, le degré moyen de la racine d'un diamants de $\mathcal{P}$ de taille n est :

$$\mathbb{E}[R_n] \underset{n\to\infty}{=} 2\log n + O(1).$$

Démonstration. On utilise le fait que $G' = G$, ainsi l'équation (1.5.1) se simplifie en $r''(z) = 1 + f(z)\cdot f''(z)$. Après une première intégration par partie et en utilisant $f'(z)^2 = 2\exp(f(z)) - 1 = 2f''(z) - 1$, on obtient $r'(z) = f(z)f'(z) - 2f'(z) + z + 2$. Et donc, puisque $r(0) = 0$:

$$r(z) = \frac{1}{2}f(z)^2 - 2f(z) + \frac{z^2}{2} + 2z.$$

La fonction $f(z) = -\log(1-\sin z)$ a le développement asymptotique suivant :

$$f(z) \underset{z\to\frac{\pi}{2}}{\sim} \log\left(\frac{8}{\pi}\right) - 2\log\left(1 - \frac{z}{\pi/2}\right) + O\left(\left(1-\frac{z}{\pi/2}\right)^2\right).$$

On en déduit celui de r :

$$r(z) \underset{z\to\frac{\pi}{2}}{\sim} \log\left(\frac{8}{\pi}\right)\left(\frac{1}{2}\log\left(\frac{8}{\pi}\right) - 2\right) + 2\log\left(1-\frac{z}{\pi/2}\right)^2 + O\left(\log\left(1-\frac{z}{\pi/2}\right)\right).$$

On utilise alors le théorème de transfert classique [FS09] :

$$n![z^n]f(z) = a_n \underset{n\to\infty}{\sim} \frac{2}{n}\left(\frac{2}{\pi}\right)^n n! \qquad n![z^n]r(z) \underset{n\to\infty}{\sim} \frac{4\ \log n}{n}\left(\frac{2}{\pi}\right)^n n!.$$

De ces deux extractions, on calcule directement $\mathbb{E}[R_n]$. □

1.5.4 Familles Polynomiales

Théorème 7:
Soit $\mathcal{F}$ une famille polynomiale de diamants croissants, définies par un polynôme G de degré m (cf. équation (1.4.1)). Quand n tend vers ∞, le degré moyen de la racine d'un diamants de $\mathcal{T}$ de taille n est :

$$\mathbb{E}[R_n] \underset{n\to\infty}{\sim} m.$$

Démonstration. En utilisant l'équation (1.5.1), on prouve :

$$r''(z) \underset{z\to\rho}{\sim} 1 + mf''(z).$$

Le résultat s'en déduit trivialement. □

1.5.5 Diamants plans

Théorème 8:
Soit $\mathcal{F}$ la famille des diamants généraux plans : $G(z) = \frac{1}{1-z}$. Quand n tend vers ∞, le degré moyen de la racine d'un diamants de $\mathcal{T}$ de taille n est :

$$\mathbb{E}[R_n] \underset{n\to\infty}{\sim} \frac{n}{\rho\sqrt{2\ \log n}}.$$

Démonstration. Dans la preuve du théorème 4, on a montré :

$$F(z) \underset{z\to\rho}{\sim} 1 - \sqrt{2}\rho\left(1 - \frac{z}{\rho}\right)\sqrt{\log\frac{1}{1-\frac{z}{\rho}}}.$$

Ici, l'équation (1.5.1) se simplifie en :

$$r''(z) = 1 + \frac{f(z)}{(1-f(z))^2}.$$

Donc

$$\frac{1}{(1-f(z))^2} \underset{z\to\rho}{\sim} \frac{1}{2\rho^2\left(1-\frac{z}{\rho}\right)^2\log\frac{1}{1-\frac{z}{\rho}}}.$$

Ainsi, on en déduit :

$$r''(z) \underset{z\to\rho}{\sim} 1 + \frac{1}{2\rho^2\left(1-\frac{z}{\rho}\right)^2\log\frac{1}{1-\frac{z}{\rho}}}.$$

L'extraction du coefficient de $[z^{n-2}]r''$ permet de conclure :

$$r_n \underset{n\to\infty}{\sim} \frac{n}{2\rho^2\log n}\rho^{-n+2}(n-2)!.$$ □

Informellement, ce dernier théorème suggère que le profil des diamants plans croissants est très "aplati" : ils sont très large et peu haut.

1.6 Conclusion

Dans cette section nous avons introduit le modèle de diamants croissants qui malgré sa simplicité à décrire n'est pas si simple à analyser combinatoirement.

Nous avons aussi présenté des bijections avec des objets connus dans la littérature, reliant ainsi notre étude à des travaux déjà existants : un intérêt motivant l'étude des structures croissantes.

CHAPITRE 2

ORDRES PARTIELS SÉRIE-PARALLÈLE

2.1 Introduction

Les ordres partiels Série-Parallèle sont présents dans une multitude de travaux et ont de nombreuses applications. Ils se caractérisent par leur structure inductive. La plupart des problèmes dits combinatoires sont résolus en temps polynomial et même souvent linéaire. Malgré cette apparente simplicité et leur histoire assez ancienne, l'énumération asymptotique des extensions linéaires de ces ordres n'a jamais été étudiée. C'est donc un des principaux problèmes que nous allons étudier dans ce chapitre.

Du point de vue des programmes concurrents, les ordre partiels Série-Parallèle sont des modèles réalistes. On peut les retrouver dans différents modèles de calcul parallèle, notamment le *Bulk Synchronous Parallel model* (BSP) introduit dans [Val90]. Parmi les intérêts de ce modèle de programmation parallèle on notera :

— sa grande abstraction : le modèle de calcul BSP permet de s'abstraire de la plupart des problèmes courants de la programmation concurrente et parallèle (*race condition*, *dead lock*, *live lock*, etc),

— sa facilité à être analysé statiquement : de par les contraintes qu'il impose, le modèle BSP permet des analyses statiques et des preuves de programmes quasi-automatiques et précises.

Bien sûr, la contrepartie est que ce modèle n'est pas aussi expressif que d'autres comme les algèbres de processus (CSP, π-calcul, etc).

Dans ce chapitre, nous présenterons nos résultats sur l'étude du comportement asymptotique du nombre de graphes Fork-Join. Pour ce faire nous aurons besoin d'introduire les produits ordonné et coloré. Enfin, nous finirons sur un développement autour de quelques propriétés fondamentales de ces opérateurs.

2.1.1 État de l'art

La première apparition des ordres partiels Série-Parallèle remontent à la fin des années 70 [VTL79]. Ils sont eux-même inspirés des travaux sur les circuits de résistances électriques Série-Parallèle [Duf65].

L'énumération des ordres partiels Série-Parallèle a été faite par R. Stanley dans [Sta74]. L'auteur donne les séries génératrices ordinaires et exponentielles de classes d'ordres partiels générées à partir d'un ensemble fini d'ordres et des compositions en série et parallèle. De là on peut en déduire différentes équations de récurrences sur les coefficients de ces séries ainsi que leur équivalent asymptotique, notamment pour celle des ordres partiels Série-Parallèle.

Du point de vue du dénombrement des extensions linéaires d'un ordre donné, l'article [Möh89] est, à notre connaissance, le premier à donner une formule inductive pour ce calcul. De cette formule on extrait (trivialement) un algorithme qui se trouve être de complexité (en nombre d'opérations arithmétiques) linéaire en la taille de l'ordre en entrée. Pour obtenir ce résultat, l'ordre est décomposé en un arbre de compositions série et parallèle dont l'existence provient directement de la définition de ces ordres.

Du point de vue de l'énumération de toutes les extensions linéaires d'ordres Série-Parallèle de taille n, il n'existe, à priori, pas de résultats antérieurs à nos travaux sur les diamants (cf. chapitre 1). On peut néanmoins, citer l'apparition de l'opérateur *ordinal product* dans le cadre de la théorie des espèces combinatoires de [BLL98], réutilisé dans [DPRS12]. Cet opérateur, permet de contraindre l'étiquetage de structures, à la manière du produit boîte.

2.2 Ordres Série-Parallèle et Graphes Fork-Join

Pour commencer, donnons les définitions formelles des compositions série et parallèle permettant de définir la classe des ordres Série-Parallèle.

Définition 3 (Compositions série et parallèle)**:**
Soit $(P, \preceq_P)$ et $(Q, \preceq_Q)$ deux ordres partiels disjoints, c'est-à-dire tels que $P \cap Q = \varnothing$.

— La composition parallèle $(R, \preceq_R)$ de $(P, \preceq_P)$ et $(Q, \preceq_Q)$, notée $R = P \parallel Q$, est l'ordre partiel obtenu par l'union disjointe des ensembles P et Q et telle que la relation d'ordre $\preceq_R$ sur R est définie par :

$$\forall E \in \{P, Q\}, \forall x, y \in E, x \preceq_R y \text{ ssi } x \preceq_E y.$$

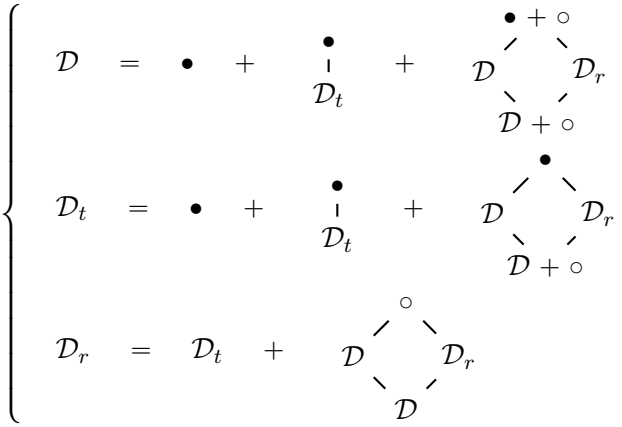
— La composition série $(S, \preceq_S)$ de $(P, \preceq_P)$ et $(Q, \preceq_Q)$, notée $R = P.Q$, est l'ordre partiel obtenu par l'union disjointe des ensembles P et Q et telle que la relation d'ordre $\preceq_S$ sur S est définie par :

$$\forall x, y \in P \cup Q, x \preceq_S y \text{ ssi } \begin{cases} x \in P \text{ et } y \in Q \\ x, y \in P \text{ et } x \preceq_P y \\ x, y \in Q \text{ et } x \preceq_Q y \end{cases}.$$

Définition 4 (Ordres partiels Série-Parallèle)**:**
La classe des ordres partiels Série-Parallèle est la plus petite classe contenant l'ordre singleton et close par compositions série et parallèle.

Comme nous l'avons précédemment fait sur les diamants, ici aussi nous ne travaillerons pas directement sur les ordres partiels mais sur les plongements combinatoires des couvertures de ces ordres, c'est-à-dire sur leur représentation graphique intransitive et planaire. De plus, principalement pour simplifier notre algorithme de génération aléatoire uniforme d'extensions linéaires, nous choisissons de travailler sur une représentation "binaire" des ordres Série-Parallèle. Pour ce faire, nous introduisons la famille des graphes Fork-Join.

Définition 5:
La classe $\mathcal{D}$ des graphes Fork-Join est définie par le système symbolique suivant :



Un graphe Fork-Join de $\mathcal{D}$ est soit un noeud isolé $\bullet$ ou bien une source (ou racine) à laquelle pend un graphe de $\mathcal{D}_t$, ou encore une source (un noeud noir $\bullet$ ou blanc $\circ$) avec un fils gauche appartenant à $\mathcal{D}$ et un fils droit appartenant à $\mathcal{D}_r$, tous les deux suivis (composition séries) par un graphe de $\mathcal{D}$ ou un noeud blanc $\circ$. Par la suite, on désignera ces différents sous-graphes par "le sous-graphe gauche, droit ou du dessous".

La classe $\mathcal{D}_t$ correspond aux couvertures connectées n'ayant qu'une seule source tandis que la classe $\mathcal{D}_r$ correspond aux couvertures connectées en général.

Théorème 9:
Les graphes Fork-Join sont en bijection avec les plongements combinatoires des couvertures d'ordres partiels Série-Parallèle.

Nous construisons la preuve de ce théorème étape par étape.

Tout d'abord nous exploitons le principe "gauchisant" (*left-leaning* en anglais). Il est utilisé dans [Sed08] afin de donner une représentation canonique d'arbres rouges-noirs. Le principe est une simple retranscription de l'associativité à gauche de l'opérateur de composition parallèle, associé au plongement combinatoire :

$$P_1 \parallel P_2 \parallel \cdots \parallel P_n \quad \text{est équivalent à} \quad (\ldots(P_1 \parallel P_2) \parallel \ldots) \parallel P_n.$$

Ce théorème permet ainsi de passer d'une représentation "n-aire" (un nombre indéfini d'ordres en parallèles) à une représentation binaire. Pour ce faire, les graphes Fork-Join sont composés

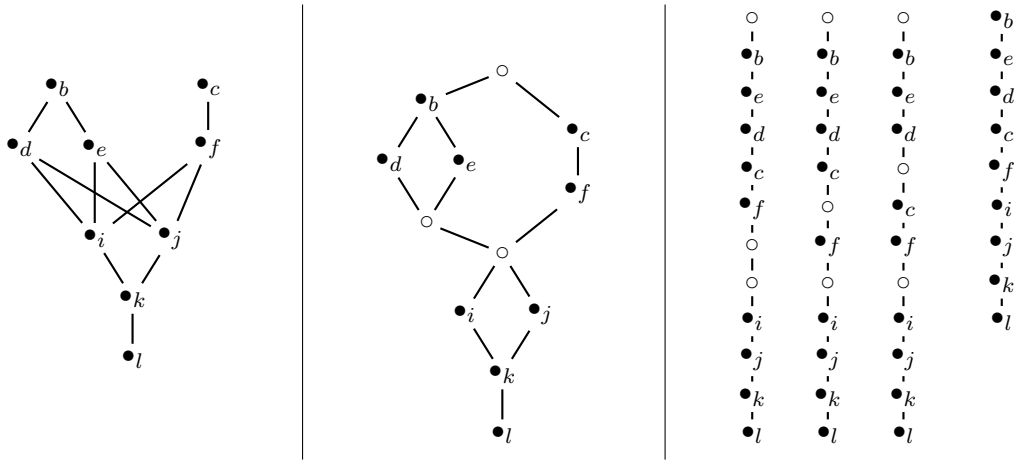


FIGURE 2.1 – (De gauche à droite) La couverture d'un ordre Série-Parallèle P ; son graphe Fork-Join correspondant ; l'ensemble des tris topologiques de la même classe d'équivalence et leur extension linéaire associée

de noeuds noirs, correspondant aux noeuds originaux de la couverture de l'ordre, et de noeuds blancs, correspondant à des noeuds ajoutés afin de connecter et "binariser" la couverture.

Par exemple, dans la figure 2.1, sur la gauche, un ordre partiel est représenté par sa couverture et au milieu on peut voir le graphe Fork-Join correspondant. On voit bien que les noeuds blancs ont été ajoutés pour remplir la contrainte binaire.

Note: Par abus de language, nous supposerons par la suite que tous les graphes (et donc les couvertures) sont combinatoirement plongés.

Définition 6:
Soit Ψ la fonction de l'ensemble des couvertures d'ordres Série-Parallèle dans l'ensemble des graphes Fork-Join, inductivement définie par :

$$\begin{cases} \Psi(\varnothing) = \circ \qquad \Psi(\bullet) = \bullet \qquad \Psi\left(\begin{array}{c}\bullet\\ \vdots\\ \bullet\end{array}\right) = \begin{array}{c}\bullet\\ \vdots\\ \bullet\end{array} \\ \Psi(P.(Q \parallel R).S) = \begin{array}{ccc} & \Psi(P) & \\ \diagup & & \diagdown \\ \Psi(Q) & & \Psi(R) \\ \diagdown & & \diagup \\ & \Psi(S) & \end{array}, \end{cases}$$

où P, Q, R et S sont des couvertures d'ordres Série-Parallèle. P et S peuvent être vides. Q et R ne sont pas vides et R vérifie $\neg(\exists R_1, R_2,\ R = R_1 \parallel R_2)$.

Quand l'ordre ressemble à $P_1 \parallel P_2 \parallel \cdots \parallel P_n$, la condition sur R dans la dernière règle signifie que $R = P_n$. Notons que la seule contrainte sur P et S est que ce sont des couvertures

Série-Parallèle pouvant être vides. Ainsi, l'application de la dernière règle est non-déterministe, dans le sens où elle peut être appliquée plusieurs fois mais dans un ordre arbitraire. Ce système de règle est trivialement confluent.

Proposition 3:
Ψ est une bijection.

Esquisse de preuve. La preuve se fait par une simple induction structurelle. Les cas concernant l'ordre vide, le singleton et la chaîne sont triviaux. Dans le dernier cas, la contrainte sur l'ordre R (qu'il n'est pas une composition parallèle de sous-ordres) garantit que le graphe obtenu aura bien la bonne forme (un profil gauchisant). Ensuite il suffit de vérifier que les noeuds blancs ne peuvent se retrouver qu'en position de source ou puits. □

La preuve du théorème 9 est donc un corollaire direct de cette proposition.

Vu que notre étude s'intéresse aux ordres partiels à travers leur ensemble d'extensions linéaires, une propriété fondamentale des graphes Fork-Join est qu'il conserve le nombre d'extensions linéaires de l'ordre initial, si l'on oublie les noeuds blancs :

— comme nous l'avons déjà énoncé, les tris topologiques d'un plongement combinatoire de la couverture sont isomorphes aux extensions linéaires de l'ordre lui même,
— dans le cas de nos graphes Fork-Join, les tris topologiques incluent les noeuds blancs, qui ne font pas partie de l'ordre initialement. On va donc les "oublier" en considérant des classes d'équivalences d'ordres topologiques sur ces graphes.

Définition 7:
Soit un ordre Série-Parallèle P et $S = \Psi(P)$ son graphe Fork-Join associé. On définit la fonction ρ, de l'ensemble des tris topologiques de S dans l'ensemble des extensions linéaires de P, qui appliquée à un tri topologique supprime tous ses noeuds blancs.

Définition 8:
Soit P un ordre Série-Parallèle et $S = \Psi(P)$ son graphe Fork-Join associé. On dit que deux tris topologiques s_1 et s_2 de S sont équivalents si et seulement si $\rho(s_1) = \rho(s_2)$.
Soit e une extension linéaire de P, on peut donc définir la classe d'équivalence de e des tris topologiques de S :

$$\{s \in S \mid \rho(s) = e\} \ .$$

Par exemple, dans la figure 2.1 sur le côté droit, nous avons représenté trois tris topologiques équivalents de l'extension linéaire $b \preceq e \preceq d \preceq c \preceq f \preceq i \preceq j \preceq k \preceq l$.

On remarquera que les extensions linéaires n'ont pas toutes le même nombre de tris équivalents. Par exemple, l'extension linéaire $c \preceq f \preceq b \preceq d \preceq e \preceq i \preceq j \preceq k \preceq l$ n'a qu'un seul tri équivalent. Cela a pour conséquence qu'un tirage aléatoire et uniforme d'un tri topologique du graphe Fork-Join $S = \Psi(P)$ ne donnera pas une extension linéaire **uniforme** de P (après application de ρ).

Définition 9:
Soit P un ordre Série-Parallèle et $S = \Psi(P)$ son graphe Fork-Join associé. Soit e une extension linéaire de P, on appelle représentant de e le tri topologique de la classe d'équivalence engendrée par e tel que

— tous les noeuds blancs apparaissent le plus tôt possible dans le tri,
— si deux noeuds blancs sont incomparables, celui le plus à gauche (dans S) apparaît en premier.

On a donc maintenant moyen de lier les extensions linéaires de P et les tris topologiques de $\Psi(P)$. Par exemple, dans la figure 2.1, sur le côté droit, des trois tris topologiques présentés, c'est le troisième qui est le représentant de l'extension linéaire de droite.

Proposition 4:
Soit P un ordre Série-Parallèle et $S = \Psi(P)$ son graphe Fork-Join associé. Une extension linéaire de P tirée aléatoirement selon la loi uniforme est obtenue par tirage aléatoire et uniforme d'un représentant dans S puis application de la fonction ρ.

Pour conclure cette section, nous discutons rapidement de la complexité de construire un graphe Fork-Join à partir d'un ordre Série-Parallèle.

Proposition 5:
Le graphe Fork-Join correspondant à un ordre Série-Parallèle sur n éléments est construit en $\mathcal{O}(n)$ opérations élémentaires (ajouts ou suppressions de noeuds ou d'arcs dans un graphe dirigé).

Le nombre de noeuds noirs du graphe Fork-Join ainsi construit est n et celui de noeuds blancs est au plus $2(n-1)$.

Démonstration. La complexité annoncé se prouve simplement par induction, en utilisant la bijection Ψ. Pour le nombre de noeuds blancs, le pire cas est l'ordre contenant n noeuds incomparables. □

2.3 Graphes Fork-Join croissants et produit ordonné

Comme dans le cas des diamants, la spécification des graphes Fork-Join est en fait très similaire à une spécification d'arbres. On cherche donc maintenant un moyen de spécifier les étiquetages croissants des noeuds noirs des graphes Fork-Join. Dans ce cas, on voit bien que l'opérateur boîte n'est pas suffisant : il ne permet de contraindre l'étiquette que d'un seul atome alors que nous voudrions ajouter une contrainte plus globale entre toutes les étiquettes des sous-graphes du dessus (gauche et droit) et celle du sous-graphe du dessous. Pour cela nous introduisons un nouvel opérateur à la méthode symbolique : le produit ordonné.

2.3.1 Définition du produit ordonné

Dans la littérature un opérateur, analogue au produit ordonné, apparaît dans le domaine de la combinatoire algébrique et plus particulièrement des espèces (cf. [BLL98, Chapitre 5]). Dans ce contexte, il est appelé *produit ordinal* (*ordinal product* en anglais). Le nom de *produit ordonné* est introduit dans l'article [DPRS12], où il est utilisé en tant que outil technique, dans le contexte de la génération à la Boltzmann de langages *quasi-réguliers*.

Définition 10:
Soient $\mathcal{A}$ et $\mathcal{B}$ deux classes combinatoires étiquetées et α et β deux structures appartenant, respectivement, à $\mathcal{A}$ et $\mathcal{B}$. On définit le produit ordonné de α et β, noté $\alpha \boxstar \beta$, comme :

$$\alpha \boxstar \beta = \left\{(\alpha,\, f_{|\alpha|}(\beta)) \mid f_{|\alpha|}(\cdot) \text{ décale les étiquettes par } |\alpha|\right\}.$$

La fonction $f_{|\alpha|}$ est une fonction de réétiquetage, décalant de $+|\alpha|$ les étiquettes de son argument.

On étend alors le produit ordonné de structures étiquetées aux classes combinatoires étiquetées :

$$\mathcal{A} \boxstar \mathcal{B} = \bigcup_{\alpha\in\mathcal{A},\ \beta\in\mathcal{B}} \alpha \boxstar \beta.$$

En fait, le produit ordonné $\mathcal{A} \boxstar \mathcal{B}$ contient les objets du produit étiqueté $A \star \mathcal{B}$ tels que toutes les étiquettes de la composante de $\mathcal{A}$ sont plus petites que celles de la composante de $\mathcal{B}$.

Remarque 3:
Dans le cas où une des opérandes du produit ordonné est réduite à la classe atomique, le produit ordonné correspond au produit boîte classique [FS09, p.139].

Dans le cadre de la méthode symbolique, on s'intéresse à la "traduction" de cet opérateur en terme d'opérations sur les séries génératrices exponentielles. Pour ce faire, on rappelle les définitions des transformées (combinatoires) de Laplace que nous noterons $\mathcal{L}_{\mathrm{c}}$ et Borel [1] que nous noterons $\mathcal{B}_{\mathrm{c}}$. D'un point de vue combinatoire, elles définissent un pont entre fonction génératrice ordinaire et exponentielle. Soit, formellement :

$$\mathcal{L}_{\mathrm{c}}\left(\sum_{n\geqslant 0} a_n \frac{z^n}{n!}\right) = \sum_{n\geqslant 0} a_n z^n; \quad \mathcal{B}_{\mathrm{c}}\left(\sum_{n\geqslant 0} a_n z^n\right) = \sum_{n\geqslant 0} a_n \frac{z^n}{n!}.$$

D'un point de vue analytique, elles correspondent aux transformations intégrales suivantes :

$$\begin{aligned}\mathcal{L}_{\mathrm{c}}(f) &= \int_0^{\infty} \exp(-t) f(zt)dt;\\ \mathcal{B}_{\mathrm{c}}(f) &= \frac{1}{2i\pi}\int_{c-i\infty}^{c+i\infty} \frac{\exp(zt)}{t} f\left(\frac{1}{t}\right) dt,\end{aligned}$$

où c est une constante réelle plus grande que la partie réelle de n'importe quelle singularité de $f(1/t)/t$.

De façon analogue à la transformée de Laplace usuelle, le produit de transformée de Laplace combinatoire peut être exprimé comme un produit de convolution :

$$z \cdot \mathcal{L}_{\mathrm{c}}(f) \cdot \mathcal{L}_{\mathrm{c}}(g) = \mathcal{L}_{\mathrm{c}}\left(\int_0^z f(t)g(z-t)dt\right).$$

1. on pourra se référer à l'annexe B dans laquelle on rappelle les transformées de Laplace et Borel usuelles et leurs liens avec leur version combinatoire

où de façon équivalente :

$$\mathcal{L}_{\mathrm{c}}(f) \cdot \mathcal{L}_{\mathrm{c}}(g) = \mathcal{L}_{\mathrm{c}} \left(\int_0^z f(t)g'(z-t)dt + g(0)f(z) \right).$$

On notera $f * g$ la convolution combinatoire : $f * g = \int_0^z f(t)g'(z-t)dt + g(0)f(z)$.

Proposition 6:
Soit $\mathcal{A}$ et $\mathcal{B}$ deux classes combinatoires étiquetées. La série génératrice exponentielle $C(z)$ associée à la classe $\mathcal{C} = \mathcal{A} \boxtimes \mathcal{B}$ est donnée par les trois formulations équivalentes suivantes :

$$\begin{aligned} C(z) &= \mathcal{B}_{\mathrm{c}} \left(\mathcal{L}_{\mathrm{c}}\, A(z) \cdot \mathcal{L}_{\mathrm{c}}\, B(z) \right) \\ &= \sum_{n \geqslant 0} \frac{\sum_{k=0}^{n} a_k b_{n-k}}{n!} z^n \\ &= A(z) * B(z). \end{aligned}$$

Maintenant le produit ordonné défini dans le cadre de la méthode symbolique, explorons quelques unes de ses propriétés.

2.3.2 Graphes Fork-Join croissants

Maintenant ces nouveaux outils présentés, nous pouvons nous recentrer sur le sujet principal de ce chapitre : les graphes Fork-Join croissants.

Pour simplifier les choses, nous ne considérerons qu'une sous-classe de graphes Fork-Join : les graphes Fork-Join monochromes. Dans cette sous-classe, les noeuds sont tous noirs. Cette approche permet de travailler avec une série génératrice monovariée, au lieu de bivariée. On définit donc les graphes Fork-Join monochromes et croissants par la spécification suivante :

$$\mathcal{F} = \mathcal{Z} + \mathcal{Z}^{\square} \star \mathcal{F} + \mathcal{Z}^{\square} \star (\mathcal{F}^2 \boxtimes \mathcal{F}). \tag{2.3.1}$$

On notera que la spécification non-étiquetée de ces graphes est similaire à une spécification d'arbres unaire-ternaires :

$$\mathcal{T} = \mathcal{Z} + \mathcal{Z} \times \mathcal{T} + \mathcal{Z} \times \mathcal{T}^3. \tag{2.3.2}$$

Dans cette section, nous présenterons un premier résultat sur l'asymptotique du nombre de graphes Fork-Join monochromes de taille n. Pour le moment, nous sommes capable d'obtenir ce résultat seulement grâce à des méthodes calculatoires. Il serait préférable d'utiliser des méthodes analytiques, généralisables à d'autre cas d'utilisation du produit ordonné. Pour le moment cela reste hors de notre de portée vu la forme complexe de l'équation vérifiée par la série génératrice.

Graphes Fork-Join monochromes Avant d'entrer dans le vif du sujet à propos des graphes Fork-Join croissants et monochromes, nous donnons un résultat classique concernant l'énumération des graphes Fork-Join monochromes non-étiquetés.

Théorème 10:
Soit T la série génératrice des graphes Fork-Join monochromes définis par la spécification (2.3.2). Alors, on a :

$$[z^n]T(z) \underset{n\to\infty}{\sim} \frac{1}{2}\sqrt{\frac{1}{3}+\frac{1}{2^{2/3}}}\frac{\tau^{-n}}{\sqrt{\pi n^3}},$$

avec $\tau = \left(1+3\cdot 2^{-\frac{2}{3}}\right)^{-1}$ le rayon de convergence de T.

La preuve de ce théorème est standard. Elle repose sur l'analyse de singularités de fonctions algébriques, théorie détaillée dans [FS09].

De la spécification (2.3.1) on déduit, grâce à la méthode symbolique, une équation différentielle et intégrale dont une solution est la fonction génératrice F des graphes Fork-Join croissants et monochromes :

$$F'(z) = 1 + F(z) + \int_0^z F'(z-t)\cdot F^2(t)\mathrm{d}t. \tag{2.3.3}$$

D'un point de vue analytique, nous n'avons pas encore réussi à caractériser cette fonction génératrice : quel est le type de sa singularité ? est-elle algébrique ou holonome ? etc.
Pour le moment, la seule information que nous avons est donnée par le lemme suivant.

Lemme 3:
F a une singularité dominante ρ réelle et strictement positive.

Démonstration. Le fait que la singularité soit réelle et positive est obtenu par application du théorème de Pringsheim (cf. [FS09, page 240]). La localisation de cette singularité est obtenue par un argument combinatoire simple : le nombre de graphes Fork-Join croissants de taille n est borné supérieurement par le nombre de graphes Fork-Join non-étiquetés de taille n multiplié par le nombre de permutations de taille n (c-à-d $n!$) ; on a donc que $[z^n]F(z) = \mathcal{O}\left(n^{\frac{3}{2}}\tau^{-n}\right)$ (F est une série exponentielle) et donc que le rayon de convergence de F est plus grand ou égal à $\tau > 0$. □

Néanmoins, malgré cette difficulté, nous sommes capables de calculer un équivalent asymptotique du nombre de graphes Fork-Join monochromes et croissants, de taille n.

Théorème 11:
Le nombre f_n de graphes Fork-Join monochromes de taille n vérifie, asymptotiquement :

$$f_n \sim 6n\ n!\ \rho^{-n-2}$$

où ρ est le rayon de convergence de F

Démonstration. En partant de l'équation différentielle (2.3.3) vérifiée par F, on en déduit l'équation de récurrence vérifiée par le terme général $(f_n)_{n\geqslant 0}$ de F :

$$f_n = f_{n-1} + \sum_{k=2}^{n-2} f_{n-1-k} \sum_{i=1}^{k-1} \binom{k}{i} f_i f_{k-i}. \tag{2.3.4}$$

On définit la suite $(c_n)_{n\geqslant 0}$ définie par : $\forall n,\ c_n = \frac{f_n}{n!}$.
De cette définition et de la relation de récurrence sur les f_n, on en déduit la relation de récurrence suivante, pour les c_n :

$$c_n = \frac{1}{n}\, c_{n-1} + \frac{1}{n} \sum_{k=2}^{n-2} c_{n-1-k}\, \frac{\sum_{i=1}^{k-1} c_i\, c_{k-i}}{\binom{n-1}{k}}. \tag{2.3.5}$$

Pour faciliter les prochaines manipulations on réécrit cette équation ainsi :

$$c_n = \frac{1}{n}\, c_{n-1} + \frac{1}{n} \sum_{\substack{i+j+k=n-1 \\ i,j,k\geqslant 1}} \frac{c_i\, c_j\, c_k}{\binom{n-1}{i+j}}. \tag{2.3.6}$$

Par la suite, on exploitera le fait que i et j sont symétriques et l'égalité $\binom{n-1}{i+j} = \binom{n-1}{n-1-k}$.

Pour prouver l'équivalent asymptotique annoncé, on va procéder par encadrement. Et pour prouver cet encadrement on utilisera une preuve par induction. Or nous sommes confrontés à deux problèmes dans ce raisonnement par induction :

— il ne fonctionne qu'à partir d'un certain rang,
— nous n'avons pas d'expression close pour ρ.

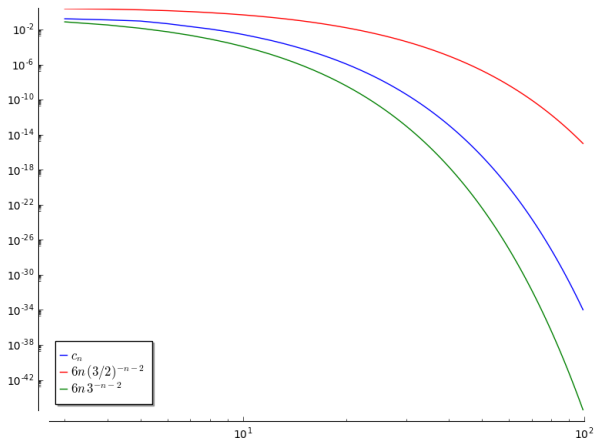
Dans la suite, nous montrons donc le principe général de la preuve qui peut être "affinée" au sens de :

$$\forall \epsilon > 0,\ \exists n_0,\ \forall n > n_0,\ \ 6n\ n!(\rho - \epsilon)^{-n-2} \leqslant f_n \leqslant 6n\ n!(\rho + \epsilon)^{-n-2}$$

Dans notre cas, nous considérerons $n_0 = 2$ et $\rho \in [\frac{3}{2}, 3]$ (numériquement on peut calculer $\rho = 2.31198067\ldots$). On veut donc prouver, pour n plus grand ou égal à 3 :

$$6n\ 3^{-n-2} \leqslant c_n \leqslant 6n\ \left(\frac{3}{2}\right)^{-n-2}.$$

On va démontrer cette inégalité par récurrence. On commence donc par montrer qu'elle est vérifiée pour au moins un entier n plus grand ou égal à 3. On obtient le graphique suivant pour les termes de 3 à 100 (en échelle *loglog*) :



La preuve étant initialisée pour un certain entier n, passons à la preuve de l'hypothèse de récurrence. On ne fera le détail que pour l'inégalité de droite, celle de gauche étant totalement symétrique.

Pour ce faire, on réécrit l'équation (2.3.6) en remplaçant c_0, c_1 et c_2 par leurs valeurs respectives 0, 1 et $\frac{1}{2}$ et en tenant compte des symétries entre i et j :

$$
\begin{aligned}
c_n = \frac{1}{n}\, c_{n-1} + \frac{1}{n}\Bigg[& \frac{c_{n-5}}{4\binom{n-1}{4}} && (1)\\
& \frac{c_{n-4}}{\binom{n-1}{3}} + \sum_{j=3}^{n-6} \frac{c_j\, c_{n-3-j}}{\binom{n-1}{j+2}} + && (2)\\
& \frac{c_{n-4}}{\binom{n-1}{n-3}} + \frac{c_{n-5}}{2\binom{n-1}{n-3}} + \frac{c_{n-3}}{\binom{n-1}{2}} + 2\sum_{j=3}^{n-5} \frac{c_j\, c_{n-2-j}}{\binom{n-1}{j+1}} + && (3)\\
& \frac{c_{n-4}}{\binom{n-1}{n-2}} + 2\frac{c_{n-3}}{\binom{n-1}{n-2}} + \sum_{\substack{i+j+k=n-1\\ i,j,k>2}} \frac{c_i\, c_j\, c_k}{\binom{n-1}{n-1-k}} + \mathcal{O}(1)\cdot \mathbb{1}_{\{n\leqslant 7\}} + && (4)\\
& \frac{1}{2\binom{n-1}{n-3}} \sum_{j=3}^{n-6} c_j\, c_{n-3-j} + && (5)\\
& \frac{1}{\binom{n-1}{n-2}} \sum_{j=3}^{n-5} c_j\, c_{n-2-j} \Bigg] && (6)
\end{aligned}
$$

Par hypothèse de récurrence forte, on a

$$\forall k,\, k \leqslant n-1 \Rightarrow c_k \leqslant 6k \left(\frac{3}{2}\right)^{-k-2}$$

En injectant cette hypothèse dans la formule précédente, on obtient une majoration de c_n. Alors, on observe que les différents termes de la somme (entre crochets) ont les ordres de grandeurs suivants [2] :

— $\mathcal{O}(n^{-3})$ pour le terme de la ligne (1)
— $\mathcal{O}(n^{-2})$ pour le terme de la ligne (2)
— $\mathcal{O}(n^{-1})$ pour le terme de la ligne (3)
— $\mathcal{O}(1)$ pour le terme de la ligne (4) (les termes contenus dans le $\mathcal{O}(1)$ ne sont présents que pour des valeurs de n entre 3 et 7)
— $\mathcal{O}(n)$ pour le terme de la ligne (5)
— $\mathcal{O}(n^2)$ pour le terme de la ligne (6)

Une fois tous ces termes multipliés par le facteur $\frac{1}{n}$ devant le crochet on a donc que seuls les termes des deux dernières lignes comptent : les autres sont de l'ordre de $\mathcal{O}\left(\frac{1}{n}\right)$.
On somme donc les termes non négligeables (les deux dernières lignes et le terme correspondant

2. On rappelle l'identité $\sum_{k=0}^{n} k\cdot(n-k) = \frac{1}{6}(n^3-n)$.

à c_{n-1}) et on obtient, en posant $\mathrm{r}(n) = 6\ n\ 3^{-n-2}$:

$$\begin{aligned}
&\frac{\mathrm{r}(n-1)}{n} + \frac{1}{2n\binom{n-1}{n-3}} \sum_{j=3}^{n-6} \mathrm{r}(j) \cdot \mathrm{r}(n-3-j) + \frac{1}{n\binom{n-1}{n-2}} \sum_{j=3}^{n-5} \mathrm{r}(j) \cdot \mathrm{r}(n-2-j) \\
&= 3^{-n-2} \frac{6\left(n^2 - 6\,n - 25\right)}{n-1} + \frac{9\left(n^2 - 9\,n - 10\right)}{(n-1)(n-2)} - \frac{18}{n} \\
&+ \frac{756}{(n-1)n} + \frac{1296}{(n-1)(n-2)n} + 18
\end{aligned}$$

En pratiquant une simple analyse de fonction, on montre que cette dernière expression est toujours inférieure à $6\ n\ \left(\frac{3}{2}\right)^{-n-2} - 3$ et donc on en conclut, par induction, que $c_n \leqslant 6\ n\ \left(\frac{3}{2}\right)^{-n-2}$.

En fait, on peut voir que les valeurs choisies pour borner ρ peuvent être aussi proches de ρ que l'on veut, du moment que l'on initialise la preuve en prenant un n_0 assez grand. Il faut alors découper la somme en morceaux contenant les termes $(c_n)_{\{n \leqslant n_0\}}$. En le faisant on observerait le même genre de découpage avec un terme dominant de la forme $\frac{1}{n\binom{n-1}{n-2}} \sum_{j=n_0+1}^{n-2n_0-1} c_j\ c_{n-n_0-j}$ qui serait encore équivalent à $6\ n\ (\rho \pm \epsilon)^{-n-2}$. □

De ce théorème, on obtient une réponse à propos de *l'explosion combinatoire* dans le cas des programmes Série-Parallèle : comme pour les ordres arborescents (cf. [BGP16]), on peut calculer l'asymptotique de la moyenne du nombre d'exécutions des programmes Série-Parallèle.

Corollaire 1:
Le nombre moyen d'exécutions des programmes Série-Parallèle de taille n quand n tend vers l'infini est équivalent à :

$$\frac{12\sqrt{\pi}}{\rho^2\sqrt{3^{-1} + 2^{-\frac{2}{3}}}} n^{\frac{5}{2}}\ n! \left(\frac{\tau}{\rho}\right)^n .$$

En remplaçant par des valeurs numériques :

$$(4.054230\ldots) \cdot (0.1496703\ldots)^n\ n^{\frac{5}{2}}\ n!.$$

2.4 Graphes Fork-Joins croissants de profondeur fixée et produit coloré

Soient deux séries génératrices ordinaires A et B, nous nous posons la question des objets énumérés par la série $\mathcal{L}_{\mathrm{c}}(\mathcal{B}_{\mathrm{c}}(A) \cdot \mathcal{B}_{\mathrm{c}}(B))$, c'est-à-dire en quelque sorte le dual du produit ordonné. En fait cette série est celle d'un opérateur sur les classes combinatoires que nous nommons le *produit coloré*. Cet opérateur nous permet d'étudier une classe intéressante de graphes Fork-Join de profondeur fixée.

2.4.1 Définition du produit coloré

Définition 11 (produit coloré)**:**
Soient $\mathcal{A}$ et $\mathcal{B}$ deux classes combinatoires, non-étiquetées et α et β deux objets respectivement de $\mathcal{A}$ et $\mathcal{B}$. On définit la classe combinatoire non-étiquetée du produit coloré de α et β, noté $\alpha \circledcirc \beta$ comme :

$$\alpha \circledcirc \beta = \left\{ (\tilde{\alpha}, \tilde{\beta}) \text{ satisfaisant la condition (C)} \right\},$$

(C) : $\tilde{\alpha}$ et $\tilde{\beta}$ ont respectivement les structures de α et β et leurs atomes sont coloriés par deux couleurs, avec la contrainte que parmi les $|\alpha| + |\beta|$ atomes de $(\tilde{\alpha}, \tilde{\beta})$, $|\alpha|$ atomes sont coloriés avec la première couleur et donc, $|\beta|$ atomes sont coloriés avec la deuxième.
On étend, naturellement, ce produit coloré de structures à celui de classe combinatoire :

$$\mathcal{A} \circledcirc \mathcal{B} = \bigcup_{\alpha \in \mathcal{A},\ \beta \in \mathcal{B}} \alpha \circledcirc \beta.$$

Remarque 4:
Dans le cas où une des opérandes du produit coloré est la classe atomique, alors le produit coloré est relié à l'opérateur de pointage (qui consiste à marquer un noeud [FS09, p.86]) :

$$\mathcal{Z} \circledcirc \mathcal{A} = \Theta(\mathcal{Z} \times \mathcal{A})$$

Proposition 7:
Soient $\mathcal{A}$ et $\mathcal{B}$ deux classes combinatoire non-étiquetées. La fonction génératrice $C(z)$, associé à la classe $\mathcal{C} = \mathcal{A} \circledcirc \mathcal{B}$, satisfait l'équation suivante :

$$C(z) = \mathcal{L}_{\mathrm{c}}\left(\mathcal{B}_{\mathrm{c}}\, A(z) \cdot \mathcal{B}_{\mathrm{c}}\, B(z)\right) = \sum_{n \geqslant 0} \sum_{k=0}^{n} \binom{n}{k} a_k b_{n-k} z^n.$$

Il est intéressant de constater que la "dualité" entre produit ordonné et produit coloré, au niveau analytique (échange des rôles entre transformée de Laplace et transformée de Borel) s'interprète aussi à un niveau combinatoire. Dans [DPRS12], le produit ordonné est utilisé comme analogue étiqueté du produit non-étiqueté pour spécifier l'opération de concaténation de langages *quasi-réguliers*. Ici, le produit coloré peut au contraire être vu comme un analogue non-étiqueté du produit étiqueté pour exprimer le mélange (*shuffle*) de langages.

Une étude plus détaillée de ces interactions entre *produits* ainsi que la construction d'opérateurs itérés à base de produit coloré sont des questions ouvertes à étudier lors de futurs travaux.

Proposition 8:
Soit $\mathcal{A}$ une classe combinatoire non-étiquetée. La série génératrice ordinaire C associée à la classe $\mathcal{A} \circledcirc \textsc{Seq}\ \mathcal{Z}$ est similaire à la *classique* transformée binomiale [Knu98] de A.

$$C(z) = \sum_{n} \sum_{k=0}^{n} \binom{n}{k} a_k = \frac{1}{1-z} A\left(\frac{z}{1-z}\right).$$

Comme dans le cas du produit ordonné, on peut énoncer plusieurs théorèmes de fermetures pour le produit coloré.

Proposition 9:
Soient A et B deux séries génératrices ordinaires et holonomes. $A \circledcirc B$ est holonome.

Proposition 10:
Soient A et B deux fonctions génératrice ordinaires et rationnelles. Alors, la série génératrice $A(z) \circledcirc B(z)$ est rationnelle.

La preuve est simple et repose sur la décomposition en éléments simples des fractions rationnelles et la formule suivante $\mathcal{L}_{\mathrm{c}}(\frac{1}{1-a\cdot z}) = e^{a\cdot z}$.

2.4.2 Graphes Fork-Join de profondeur parallèle fixée

La profondeur parallèle est un paramètre inductif des graphes Fork-Join. Du point de vue des programmes concurrents, c'est le nombre de maximal de fils d'exécution pouvant s'exécuter au même moment. Avec un vocabulaire de théorie des ordres, c'est le logarithme en base 2 de la largeur de l'ordre ou la longueur de la plus longue antichaîne (la largeur d'un ordre Fork-Join monochromes étant une puissance de 2). Une autre définition est donnée ci-dessous, en utilisant une spécification combinatoire.

Définition 12:
On note $\mathcal{N}_\ell$ la classe combinatoire des graphes Fork-Join croissants de profondeur parallèle au plus ℓ. La suite $\mathcal{N}_\ell$ est définie inductivement par le système suivant :

$$\begin{aligned}
\mathcal{N}_0 &= \underset{\geqslant 1}{\textsc{Seq}}\ \mathcal{Z},\\
\mathcal{N}_\ell &= \mathcal{Z} + \mathcal{Z}^{\,\square} \star \mathcal{N}_\ell + \mathcal{Z}^{\,\square} \star \left((\mathcal{N}_{\ell-1} \star \mathcal{N}_{\ell-1})\ \boxed{\star}\,\mathcal{N}_\ell\right).
\end{aligned}$$

Il est évident qu'un graphes Fork-Join de N_ℓ a donc au plus 2^ℓ fils d'exécutions en parallèle. En utilisant la méthode symbolique on obtient :

$$\begin{aligned}
N_0(z) &= \frac{z}{1-z},\\
N'_\ell(z) &= 1 + N_\ell(z) + \mathcal{B}_{\mathrm{c}}[\mathcal{L}_{\mathrm{c}}(N_{\ell-1}(z)) \times \mathcal{L}_{\mathrm{c}}(N_{\ell-1}(z))].
\end{aligned}$$

Soit, après une application de la transformée de Laplace et un peu de réécriture :

$$\begin{aligned}
\mathcal{L}_{\mathrm{c}}(N_0(z)) &= \frac{z}{1-z},\\
\mathcal{L}_{\mathrm{c}}(N_\ell(z)) &= \frac{z}{1 - z - z\,\mathcal{L}_{\mathrm{c}}(N_{\ell-1}(z)) \circledcirc \mathcal{L}_{\mathrm{c}}(N_{\ell-1}(z))}.
\end{aligned}$$

En utilisant la proposition 10 et un raisonnement par récurrence, on obtient que $\mathcal{N}_\ell$ est rationnel et donc holonome. De plus, on peut calculer les premiers termes de cette suite de série. Par récurrence on peut prouver que les degrés des polynômes au numérateur et au dénominateur sont égaux et régis par la récurrence suivante :

$$d_1 = 3; \qquad d_\ell = \frac{(d_{\ell-1}+1)(d_{\ell-1}+2)}{2}.$$

Les premiers termes sont 3, 10, 66, 2278, ... En conséquence, il est difficile d'obtenir des résultats quantitatifs pour les classes après $\mathcal{N}_4$, mais on peut simplement obtenir "totalement" la combinatoire des classes $\mathcal{N}_{\leqslant 4}$.

2.5 Produits ordonné et coloré : propriétés fondamentales et exemples

Les opérateurs de produits ordonné et coloré que nous avons développés pour les graphes Fork-Join sont en fait également intéressant pour d'autres types de spécifications. Dans cette section, nous établissons des propriétés fondamentales sur le produit ordonné dont nous n'avons pas eue besoin pour les graphes Fork-Join mais qui sont intéressantes d'un point de vue plus général. Nous donnons également des exemples d'utilisation de ces produits.

2.5.1 Quelques propriétés du produit ordonné

Proposition 11:
Le produit étiqueté est associatif et "commutatif", dans le sens où la convolution associée l'est et on peut trivialement associer un objet de $\mathcal{A} \boxstar \mathcal{B}$ à un de $\mathcal{B} \boxstar \mathcal{A}$.

De plus, il se distribue par rapport à l'union disjointe (ou addition sur les séries).

Proposition 12:
Soient $\mathcal{A}$ et $\mathcal{B}$ deux classes combinatoires étiquetées :

$$\begin{aligned}(A(z) * B(z))' &= A(z) * B'(z) + A'(z)B(0)\\ &= A'(z) * B(z) + B'(z)A(0).\end{aligned}$$

On peut donner une interprétation combinatoire de ce résultat en s'inspirant de la théorie des espèces [BLL98]. On interprète l'opérateur de différentiation sur les séries comme celui correspondant à la suppression d'un atome de plus petite (première identité) ou plus grande (deuxième identité) étiquette.
Par exemple, pour la première identité, la série $(A(z) * B(z))'$ correspond au produit ordonné de $\mathcal{A}$ et $\mathcal{B}$ dans lequel on a supprimé les atomes de plus grande étiquette à chaque élément. Étant donné que ces atomes appartiennent nécessairement aux objets de $\mathcal{B}$ (sauf dans le cas où $\mathcal{B}$ contient des objets de taille 0), cela est équivalent à supprimer tous les atomes de plus grande étiquette de $\mathcal{B}$, puis de faire le produit ordonné avec $\mathcal{A}$.

Avant de d'énoncer quelques propositions sur la fermeture de certaines classes de séries génératrices par produit ordonné, on rappelle la définition des séries holonomes (ou D-finies).

Définition 13:
Une série génératrice $F(z)$ est dite holonome (ou D-finie) si elle est solution d'une équation différentielle dont les coefficients sont des polynômes (où de manière équivalente, des quotients de polynômes) en z :

$$q_0(z)F(z) + q_1(z)F^{(1)}(z) + \cdots + q_r(z)F^{(r)}(z) = 0,$$

pour de quelconques polynômes q_s (avec q_r différent de 0), avec $F^{(i)}(z) = \dfrac{\mathrm{d}^i}{\mathrm{d}z^i}F(z)$.

Proposition 13:
Soient A et B deux séries génératrices exponentielles et holonomes. $A * B$ est holonome.

Démonstration. La preuve repose sur l'équivalence entre séries holonomes et suites P-récurrentes. Il est simple de montrer que l'ensemble des suites P-récurrentes est clos par transformées de Laplace et Borel et donc par produit ordonné. Il s'en suit directement le même résultat sur les séries. □

Une autre classe de séries qu'il est commun de rencontrer en combinatoire est la classe des séries ou fonctions rationnelles.

Définition 14:
Soient P et Q deux polynômes, avec $Q \neq 0$, alors $F = P/Q$ est une fonction rationnelle.

Proposition 14:
L'ensemble des fonctions rationnelles **n'est pas** clos par convolution combinatoire

Démonstration. Soient $A(z) = z$ et $B(z) = 1/(1-z)$, un calcul simple donne $A(z) * B(z) = -\log(1-z)$ qui n'est pas rationnelle. □

On se base maintenant sur le produit ordonné pour construire les opérateurs de produits itérés SEQ, CYC et SET.

2.5.2 Opérateurs de produits itérés

Commençons par définir l'exponentiation ordonnée d'une classe combinatoire étiquetée : $\mathcal{A}^{\boxed{\star}\,k} = \underbrace{\mathcal{A} \boxed{\star} \ \ldots \ \boxed{\star}\, \mathcal{A}}_{k \text{ fois}}$. On prendra la convention $\mathcal{A}^{\boxed{\star}\,0} = \mathcal{E} = \{\epsilon\}$.

Donc, un objet de $\mathcal{A}^{\boxed{\star}\,k}$ est un k-uplet où les atomes de chaque composante sont étiquetés par tous les entiers d'un intervalle de $\mathbb{N}\backslash\{0\}$, tous ces intervalles sont disjoints deux à deux et leur union est un intervalle d'entier commençant à 1. De plus, par définition du produit ordonné, on a que pour tout k-uplet $\{\alpha_1, \ldots, \alpha_k\}$ de $\mathcal{A}^{\boxed{\star}\,k}$ et tout couple d'entier i et j tel que $i < j$, les étiquettes de l'objet α_i sont plus petites que celles de l'objet α_j.

Cette propriété nous permet d'interpréter les k-uplets de $\mathcal{A}^{\boxed{\star}\,k}$ comme des représentations canoniques des **ensembles** de k objets de $\mathcal{A}$ étiquetés par des intervalles disjoints d'entiers. On utilise donc ces k-uplets pour construire des ensembles ordonnés.

Définition 15:
L'ensemble ordonné $\textsc{Set}^{\boxed{\star}}(\mathcal{A})$ d'une classe combinatoire étiquetée $\mathcal{A}$ ne contenant pas d'objets de taille nulle est défini par :

$$\textsc{Set}^{\boxed{\star}}(\mathcal{A}) = \bigcup_{k \geqslant 0} \mathcal{A}^{\boxed{\star}\,k}.$$

Proposition 15:
La série génératrice exponentielle S associée à $\textsc{Set}^{\boxed{\star}}(\mathcal{A})$ vérifie :

$$\begin{aligned} S(z) &= \sum_{n \geqslant 0} A(z)^{*n} \\ &= \mathcal{B}_{\mathrm{c}}\left(\frac{1}{1 - \mathcal{L}_{\mathrm{c}}(A(z))}\right), \end{aligned}$$

où A^{*k} signifie $\underbrace{A * \cdots * A}_{k \text{ fois}}$.

La preuve de cette proposition est une conséquence directe de la définition du produit ordonné et de la linéarité des transformées de Laplace et Borel.

On continue en définissant la séquence ordonnée. Dans les cas étiquetés et non-étiquetés, l'opérateur SEQ peut être vu comme la linéarisation des opérateurs SET et POWERSET. Ici, on prend le même point de vue.

Définition 16:
La séquence ordonnée $\textsc{Seq}^{\boxed{*}}(\mathcal{A})$ d'une classe combinatoire étiquetée $\mathcal{A}$ ne contenant pas d'objets de taille nulle est défini par :

$$\textsc{Seq}^{\boxed{*}}(\mathcal{A}) = \bigcup_{k \geqslant 0} \bigcup_{\sigma \in \mathcal{S}_k} \{\sigma(\boldsymbol{\alpha}) \mid \forall \boldsymbol{\alpha} \in \mathcal{A}^{\boxed{*}\,k}\},$$

où $\mathcal{S}_k$ désigne l'ensemble des permutations de taille k.

Proposition 16:
La série génératrice exponentielle S associée à $\textsc{Seq}^{\boxed{*}}(\mathcal{A})$ vérifie :

$$\textsc{Seq}^{\boxed{*}}(A(z)) = \sum_{k \geqslant 0} k! \cdot A(z)^{\boxed{*}\,k} = \mathcal{B}_{\mathrm{c}}(\mathcal{L}_{\mathrm{c}}[\frac{1}{1-u}](\mathcal{L}_{\mathrm{c}}(A(z)))).$$

où $\mathcal{L}_{\mathrm{c}}[\frac{1}{1-u}](F(z))$ correspond à la composition de la transformée de Laplace de $u \to \frac{1}{1-u}$ et F.

Enfin, toujours comme dans les cas étiqueté et non-étiqueté, on définit l'opérateur CYC comme le quotient de l'opérateur séquence (ordonnée dans ce cas) par la relation d'équivalence entre k-uplets dont l'un est obtenu par permutation cyclique des composantes de l'autre.

Définition 17:
Le cycle ordonné $\textsc{Cyc}^{\boxed{*}}(\mathcal{A})$ d'une classe combinatoire étiquetée $\mathcal{A}$ ne contenant pas d'objets de taille nulle est défini par :

$$\textsc{Cyc}^{\boxed{*}}(\mathcal{A}) = \bigcup_{k \geqslant 0} \bigcup_{\sigma \in S_k \backslash \mathsf{shift}_k} \{\sigma(\boldsymbol{\alpha}) \mid \forall \boldsymbol{\alpha} \in \mathcal{A}^{\boxed{*}\,k}\},$$

où $\mathcal{S}_k \backslash \mathsf{shift}_k$ est l'ensemble des permutations de taille k quotienté par la fonction $\mathsf{shift}_k(i) = i \bmod k$.

Proposition 17:
La série génératrice exponentielle C associée à $\textsc{Cyc}^{\boxed{*}}(\mathcal{A})$ vérifie :

$$\begin{aligned}
\textsc{Cyc}^{\boxed{*}}(A(z)) &= \sum_{k \geqslant 1} (k-1)! \cdot A(z)^{\boxed{*}\,k} \\
&= \mathcal{B}_{\mathrm{c}}\left(\mathcal{L}_{\mathrm{c}}(A(z)) \cdot \sum_{k \geqslant 0} k! \;\; \mathcal{L}_{\mathrm{c}}(A(z))^k\right) \\
&= A(z) * \textsc{Seq}^{\boxed{*}}(A(z)).
\end{aligned}$$

Plusieurs exemples d'utilisation de ces opérateurs seront présentés dans la section 2.5.4.

2.5.3 Théorèmes de transferts

Dans le cadre de la combinatoire analytique, la question naturelle suivant l'introduction d'un nouvel opérateur dans la méthode symbolique concerne les propriétés analytiques de celui-ci. À la manière de [FO90], on introduit un nouveau théorème de transfert pour le produit ordonné et un pour l'ensemble ordonné. Un théorème de transfert permet de faire le lien entre le comportement d'une série génératrice à l'approche de sa singularité et le comportement asymptotique du terme général de cette série : il *transfère* les connaissances du monde analytique au monde combinatoire.

Le premier théorème que nous présentons concerne le produit ordonné et est analogue au théorème de transfert de [Ben75] dans le contexte des séries génératrices à croissance rapide.

Théorème 12:
Soient A et B deux séries génératrices exponentielles. On note a (resp. b) la valuation de A (resp. B). Si il existe un entier r tel que $\sum_{k=r}^{n-r} A_k B_{n-k} = \mathcal{O}(A_{n-r} + B_{n-r})$ alors

$$[z^n]A(z) * B(z) \underset{n\to\infty}{\sim} B_b A_{n-b} + A_a B_{n-a}.$$

Ce théorème permet donc d'obtenir le premier ordre du comportement asymptotique. Notons que dans de nombreux cas $B_b A_{n-b}$ et $A_a B_{n-a}$ ne sont pas du même ordre : les singularités dominante de A et B sont différentes, a et b sont distincts ou encore, les facteurs sous-exponentiels dans le comportement asymptotique de A_n et B_n sont différents.

Démonstration. Démontrons ce théorème dans le cas où les rayons de convergence de A et B sont les même (on le note ρ) et sont dans l'intervalle $]0, +\infty[$. Les autres cas sont plus simples et reposent sur une légère adaptation de cette preuve.

Pour tout $\epsilon > 0$, il existe un entier n_0 tel que pour tout entier n plus grand que n_0, on a :

$$\frac{1-\epsilon}{\rho} \leqslant \frac{A_n}{nA_{n-1}} \leqslant \frac{1+\epsilon}{\rho} \qquad \text{et} \qquad \frac{1-\epsilon}{\rho} \leqslant \frac{B_n}{nB_{n-1}} \leqslant \frac{1+\epsilon}{\rho}.$$

Ainsi, on note $R = \max\{n_0, r\}$ puis l'on décompose le terme C_n en fonction des valuations de A et B :

$$\begin{aligned} C_n &= \sum_{k=a}^{n-b} A_k B_{n-k} = A_a B_{n-a} + A_{n-b} B_b + \sum_{k=a+1}^{n-b-1} A_k B_{n-k} \\ &= A_a B_{n-a} + A_{n-b} B_b + \left(\sum_{k=a+1}^{R-1} A_k B_{n-k} + \sum_{k=n-R+1}^{n-b-1} A_k B_{n-k} + \sum_{k=R}^{n-R} A_k B_{n-k} \right). \end{aligned}$$

On constate alors que le premier terme de la première somme est négligeable devant B_{n-a} :

$$\begin{aligned} \frac{B_{n-(R-1)}}{B_{n-a}} \cdot \sum_{k=a+1}^{R-1} \frac{A_k B_{n-k}}{B_{n-(R-1)}} &= \frac{B_{n-(R-1)}}{B_{n-a}} \cdot \sum_{k=a+1}^{R-1} A_k \frac{B_{n-k}}{B_{n-(k+1)}} \frac{B_{n-(k+1)}}{B_{n-(k+2)}} \cdots \frac{B_{n-(R-2)}}{B_{n-(R-1)}} \\ &\leqslant \frac{B_{n-(R-1)}}{B_{n-a}} \cdot \sum_{k=a+1}^{R-1} A_k \left(\frac{(n-(a+1)(1+\epsilon)}{\rho} \right)^{R-1-k}. \end{aligned}$$

Donc, pour n suffisamment grand, il existe une constante γ telle que

$$\begin{aligned}\frac{B_{n-(R-1)}}{B_{n-a}} \cdot \sum_{k=a+1}^{R-1} \frac{A_k B_{n-k}}{B_{n-(R-1)}} &\leqslant \frac{B_{n-(R-1)}}{B_{n-a}} \cdot \gamma \left(\frac{(n-(a+1)(1+\epsilon)}{\rho} \right)^{R-a-2} \\ &\leqslant \frac{B_{n-(R-1)}}{B_{n-(R-2)}} \frac{B_{n-(R-2)}}{B_{n-(R-3)}} \cdots \frac{B_{n-(a+1)}}{B_{n-a}} \cdot \gamma \left(\frac{(n-(a+1)(1+\epsilon)}{\rho} \right)^{R-a-2} \\ &\leqslant \gamma \left(\frac{(n-(a+1))(1+\epsilon)}{(n-(R-2))(1-\epsilon)} \right)^{R-a-2} \cdot \frac{\rho}{(n-(R-2))(1-\epsilon)}.\end{aligned}$$

On en déduit :

$$\lim_{n\to\infty} \frac{B_{n-(R-1)}}{B_{n-a}} \cdot \sum_{k=a+1}^{R-1} \frac{A_k B_{n-k}}{B_{n-(R-1)}} = 0.$$

De la même façon, on prouve :

$$\lim_{n\to\infty} \frac{A_{n-(R-1)}}{A_{n-b}} \cdot \sum_{k=n-R+1}^{n-b-1} \frac{A_k B_{n-k}}{A_{n-(R-1)}} = 0.$$

Par hypothèse, la dernière somme satisfait $\sum_{k=R}^{n-R} A_k B_{n-k} = \mathcal{O}(A_{n-R} + B_{n-R})$, et donc le théorème est prouvé dans ce cas. □

Remarque 5:
L'hypothèse d'application du théorème 12 est généralement assez simple à démontrer. Par exemple, dans le cas standard où $A_n = \mathcal{O}\left(n^\alpha \rho_A^{-n}\, n!\right)$ et $B_n = \mathcal{O}\left(n^\beta \rho_B^{-n}\, n!\right)$ l'hypothèse est vérifiée : il existe r tel que $\sum_{k=r}^{n-r} A_k B_{n-k} = \mathcal{O}(A_{n-r} + B_{n-r})$.

On peut étendre ce théorème au cas des ensembles ordonnés.

Théorème 13:
Soient $\mathcal{A}$ une classe combinatoire étiquetée sans élément de taille nulle et S la série génératrice exponentielle de $\textsc{Set}^{\boxtimes}(\mathcal{A})$. On pose $L(z) = \mathcal{L}_c(A(z))$ et on note ρ son rayon de convergence. Alors, le comportement asymptotique de $[z^n]S(z)$ quand n tend vers l'infini est

1. $[z^n]S(z) \underset{n\to\infty}{\sim} [z^n]A(z)$, si $\rho = 0$ et $\frac{A_n}{A_{n-1}} = \Omega\left(n^\alpha\right)$, où α est une constante strictement positive.
2. $[z^n]S(z) \underset{n\to\infty}{=} \frac{1}{n!\sigma L'(\sigma)} \cdot \sigma^n(1 + C^n)$ si $\rho \in]0, +\infty[$ et $\lim_{z\to\rho} L(z) > 1$, avec C une constante réelle telle que $0 < C < 1$ et σ vérifiant $L(\sigma) = 1$.

Démonstration. 1. Dans ce cas, on procède par induction. On définit $S_k = 1 + A * S_{k-1}$ (et $S_1 = 1 + A$), qui est la série génératrice de l'ensemble ordonné contenant au plus k objets de A. Ainsi, on a $[z^n]S_1 = [z^n]A$. On pose donc comme hypothèse d'induction : il existe un entier k tel que $[z^n]S_k \sim [z^n]A$. Maintenant, prouvons que $[z^n]S_{k+1} \underset{n\to\infty}{\sim} [z^n]A$. Par définition $S_{k+1} = 1 + A * S_k$, donc $[z^n]S_{k+1} = [z^n]A * S_k$. Puisque $\frac{A_n}{A_{n-1}} = \Omega\left(n^\alpha\right)$, on peut utiliser le théorème 12 et on obtient $[z^n]S_{k+1} \underset{n\to\infty}{\sim} A_a[z^{n-a}]S_k + [z^n]A$ où

$a > 0$ est la valuation de A. De l'hypothèse d'induction, on déduit que $[z^n]S_{k+1} \underset{n\to\infty}{\sim} A_a[z^{n-a}]A + [z^n]A \underset{n\to\infty}{\sim} [z^n]A$, ce qui conclut la preuve pour ce cas.

2. Le second cas est prouvé par application directe du théorème des *asymptotiques des suites super-critiques* [FS93].

□

Théorème 14 (Asymptotiques des suites super-critiques [FS93])**:**
Soient deux séries génératrices F et G telles que $F = (1 - G)^{-1}$. Cette relation est dite super-critique si $\lim_{z\to\rho_G} G(z) > 1$. Si l'on suppose que G est fortement apériodique [3] et que la relation entre F et G est super-critique, alors

$$[z^n]F(z) = \frac{1}{\sigma G'(\sigma)} \cdot \sigma^{-n}(1 + \mathcal{O}(A^n))$$

où $\sigma \in]0, \rho_G[$ vérifie $G(\sigma) = 1$ et A appartient à $[0, 1[$.

Remarque 6:
Le premier cas du théorème s'interprète de la façon suivante : pour une classe combinatoire $\mathcal{A}$, si A_n croit assez vite alors, quand n tend vers l'infini, un ensemble ordonné d'objets de A et de taille n contient, presque sûrement, un seul objet de taille n.

2.5.4 Exemples

On conclut cette section en revisitant quelques classes combinatoires classiques pouvant être spécifiées à l'aide des opérateurs ordonnés.

Arbres binaires de recherche

Soit $\mathcal{B}$ la classe combinatoire étiquetée des arbres binaires de recherche ayant pour clé l'ensemble des entiers compris entre 1 et la taille de l'arbre. Un arbre binaire de recherche est soit une feuille contenant 1, soit un noeud interne d'étiquette i père de deux sous-arbres, tel que toutes les étiquettes du sous-arbre gauche (resp. droit) sont plus petites (resp. grandes) que i.

En utilisant le produit ordonné on peut maintenant donner une spécification de cette classe :

$$\mathcal{B} = \mathcal{E} + (\mathcal{B} \boxstar \mathcal{Z}) \boxstar \mathcal{B}.$$

Cette spécification se traduit en équation fonctionnelle

$$B(z) = 1 + (B(z) * z) * B(z).$$

Par définition du produit ordonné, et après application de la transformée de Laplace combinatoire, on obtient

$$\mathcal{L}_{\mathrm{c}}(B(z)) = 1 + \mathcal{L}_{\mathrm{c}}(B(z)) \times z \times \mathcal{L}_{\mathrm{c}}(B(z)).$$

On y reconnaît l'équation classique de la série génératrice des arbres binaires. On peut donc en déduire directement le nombre d'arbres binaires de recherche de taille n.

3. G est fortement apériodique si il n'existe pas de fonction analytique h telle que $G(z) = h(z^d)$ où $d \geqslant 2$

Compositions d'entiers

Une composition d'un entier n est une suite d'entiers strictement positifs dont la somme vaut n. Une façon de manipuler combinatoirement les compositions d'entiers est de spécifier leur classe non-étiquetée ainsi

$$\textsc{Seq}(\underset{\geqslant 1}{\textsc{Seq}}(\mathcal{Z})).$$

Même si cette spécification est valable dans les cas étiquetés ou non, elle s'interprète plus facilement dans le cas non-étiqueté. Chaque entier est représenté par une séquence d'au moins un atome ($\textsc{Seq}_{\geqslant 1}(\mathcal{Z})$) et la séquence de ces entiers représente la composition.

À l'aide de l'opérateur d'ensemble ordonné, on peut donner une spécification "plus" étiquetée des compositions d'entiers. Dans ce contexte, il nous semble plus naturelle de représenter un entier k par un ensemble de k atomes étiquetés de 1 à k. Alors, une composition d'entiers devient un ensemble ordonné d'ensembles d'atomes étiquetés :

$$\textsc{Set}^{\boxed{\star}}(\underset{\geqslant 1}{\textsc{Set}}(\mathcal{Z})).$$

Par exemple, l'ensemble $\{\{1, 2, 3\}, \{6, 5\}, \{7, 8, 9\}, \{4\}\}$ représente la composition $3 + 1 + 2 + 3$ de 9.

On peut alors travailler sur la série génératrice exponentielle donnée par

$$\mathcal{B}_{\mathrm{c}}\left(\frac{1}{1-\mathcal{L}_{\mathrm{c}}(e^z-1)}\right) = \mathcal{B}_{\mathrm{c}}\left(\frac{1-z}{1-2z}\right).$$

Notons que même si l'extraction du n-ième coefficient de cette série est simple, une application directe de notre théorème 13 donne aussi le même résultat (deuxième cas du théorème) :

$$[z^n]\,\mathcal{B}_{\mathrm{c}}\left(\frac{1-z}{1-2z}\right) = 2^{n-1}.$$

Permutations intervalles généralisées

Il est bien connu qu'une permutation peut-être décomposée de manière unique en un ensemble de cycles. De cette correspondance, on tire la spécification suivante, pour la classe étiquetée des permutations :

$$\textsc{Set}(\textsc{Cyc}(\mathcal{Z})).$$

On s'intéresse maintenant à une sous-classe de permutations introduite dans [KM12] et nommées permutations intervalles généralisées (PIG). Une PIG est une permutation dont les cycles une fois triés sont des intervalles d'entiers. Par exemple, la permutation (231489567) est une PIG car elle admet la décomposition $\{(\overrightarrow{123})(\overrightarrow{4})(\overrightarrow{58697})\}$ (où $c = \overrightarrow{a_0 \ldots a_{n-1}}$ est tel que $c(a_i) = a_{i+1 \bmod n}$).

On peut donc naturellement spécifier ces permutations à l'aide de l'ensemble ordonné :

$$\textsc{Set}^{\boxed{\star}}(\textsc{Cyc}(\mathcal{Z})).$$

Comme précédemment, on obtient simplement la série génératrice exponentielle de cette classe :

$$\mathcal{B}_{\mathrm{c}}(\frac{1}{1-\mathcal{L}_{\mathrm{c}}(\log(\frac{1}{1-z}))}).$$

On remarque alors que $\mathcal{L}_{\mathrm{c}}(\log \frac{1}{1-z}) = \sum_{n\geqslant 0} n!\, z^n$ a un rayon de convergence nul et donc que la série a un rayon de convergence nul. Mais, on peut utiliser le premier cas du théorème 13 pour directement obtenir

$$[z^n]\, \mathcal{B}_{\mathrm{c}}(\frac{1}{1-\mathcal{L}_{\mathrm{c}}(\log(\frac{1}{1-z}))}) \sim (n-1)!.$$

Ensemble ordonné de diamants croissants binaires

Pour illustrer un autre cas d'application du théorème de transfert 13 et en relation avec la concurrence, nous présentons le modèle des ensembles ordonnés de diamants croissants binaires. Le modèle de diamants binaires $\mathcal{D}$ que nous considérons est spécifié ainsi :

$$\mathcal{D} = \mathcal{Z} + \mathcal{Z} \times (\mathcal{E} + \mathcal{D} + \mathcal{D}^2) \times \mathcal{Z}$$

Donc, dans le cadre non-étiqueté, on considère la classe $\mathcal{S}$ des séquences de diamants binaires :

$$\mathcal{S} = \textsc{Seq}(\mathcal{D}) - \textsc{Seq}(\underset{\geqslant 1}{\textsc{Seq}}(\mathcal{Z})) + \textsc{Seq}(\mathcal{Z})$$

On se permet d'utiliser l'opérateur de soustraction pour supprimer des ambiguïtés : en ne considérant que $\textsc{Seq}(\mathcal{D})$, on énumère plusieurs fois les séquences d'atomes ($\textsc{Seq}(\mathcal{Z})$) qui sont déjà présentes dans $\mathcal{D}$. En supprimant $\textsc{Seq}(\textsc{Seq}_{\geqslant 1}(\mathcal{Z}))$ (que l'on reconnaît être la spécification des compositions d'entiers) on supprime toutes les séquences d'atomes construites par $\textsc{Seq}(\mathcal{D})$; puis on ajoute, une seule fois, les séquences d'atomes avec $\textsc{Seq}(\mathcal{Z})$.

En nommant $\mathcal{B}$ la classe des diamants binaires croissants et celle des ensemble ordonnés de diamants binaires croissants, on obtient les spécifications suivantes :

$$\begin{aligned}\mathcal{B} &= \mathcal{Z} + \mathcal{Z}^{\square} \star (\mathcal{E} + \mathcal{B} + \mathcal{B}^2) \star \mathcal{Z}^{\blacksquare} \\ &= \textsc{Set}^{\boxtimes}(\mathcal{B}) - \textsc{Set}^{\boxtimes}(\underset{\geqslant 1}{\textsc{Set}}(\mathcal{Z})) + \textsc{Set}(\mathcal{Z})\end{aligned}$$

On obtient alors l'équation fonctionnelle suivante pour :

$$= \mathcal{B}_{\mathrm{c}}\left(\frac{1}{1-\mathcal{L}_{\mathrm{c}}(B(z))}\right) - z \cdot e^{2z} + e^z$$

De cette équation on peut alors tirer le résultat suivant par utilisation de notre théorème de transfert 13.

Théorème 15: — L'asymptotique du nombre d'ensembles ordonnés de diamants binaires croissants de taille n est équivalent au nombre de diamants binaires croissants :

$$n!\, [z^n]O \underset{n\to\infty}{\sim} n!\, [z^n]B.$$

— L'asymptotique de la moyenne du nombre d'étiquetages croissants de séquences de diamants binaires de taille n est

$$\frac{[z^n]\, O(z)}{[z^n]\, (1 - D(z))^{-1}} \underset{n\to\infty}{\sim} 6\, D'(\sigma)\, \frac{\sigma^{n+1} \cdot (n+1)!}{\rho^{n+2}}$$

$\sigma = \frac{1}{6}(\sqrt{13} - 1)$ est une solution $D(z) = 1$, et $D'(\sigma) = \sqrt{13}/\sigma$.
On peut approximer ces constantes ainsi

$$\frac{[z^n]\, O(z)}{[z^n]\, (1 - D(z))^{-1}} \underset{n\to\infty}{\sim} \alpha \cdot \beta^{n+1} \cdot (n+1)!$$

avec $\alpha \approx 15.7042\ldots$ et $\beta \approx 0.136896\ldots$.

Démonstration. Le premier asymptotique est obtenu par application direct du théorème 13. Le terme $[z^n]e^z + ze^{2z}$ étant négligeable devant celui de $\mathcal{B}_{\mathrm{c}}\left(\frac{1}{1-\mathcal{L}_{\mathrm{c}}(B(z))}\right)$.

Le second résultat est obtenu en calculant le quotient du nombre d'ensembles ordonnés de diamants croissants par celui du nombre de séquences de diamants non-étiquetés. D peut être vue comme la série génératrice ordinaire d'une famille simplement généré d'arbres et dont l'analyse asymptotique est bien connue [FS09]. Pour calculer le nombre asymptotique de séquences de diamants de taille n, on applique le théorème *asymptotiques des suites super-critiques* [FS93]. □

Nombres de Bell

Les nombres de Bell, notés B_n, sont connus pour compter le nombre de partitions d'un ensemble de n éléments. Ils vérifient l'équation de récurrence $B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k$ (avec $B_0 = 1$). Une spécification classique de la classe étiquetée $\mathcal{B}$ des partitions d'ensembles est $\mathcal{B} = \textsc{Set}(\textsc{Set}_{\geqslant 1}\ \mathcal{Z})$. Une partition d'un ensemble est un ensemble d'ensembles tel que la somme des tailles des ensembles internes est égale à la taille de l'ensemble partitionné. De cette spécification, en appliquant la méthode symbolique, on obtient une formule close pour la série génératrice exponentielle de $\mathcal{B}$:

$$B(z) = \sum_{n\geqslant 0} B_n \frac{z^n}{n!} = e^{e^z - 1}.$$

Ici, on se propose de revisiter cette classe dans un contexte non-étiqueté, en utilisant le produit coloré. Dans ce contexte, une partition d'un ensemble de n éléments en k parts est isomorphe à un coloriage à k couleurs de ces n éléments. Donc, on peut voir cette partition comme un premier ensemble de taille p (un coloriage de p éléments avec la première couleur) qu'on ajoute à l'ensemble d'une partition des $n - p$ éléments restants (un coloriage à $k - 1$ couleurs du reste des éléments).

On peut donc traduire cette décomposition, en utilisant la méthode symbolique et le produit coloré :

$$\mathcal{B} = \mathcal{E} + \mathcal{Z} \times (\textsc{Seq}(\mathcal{Z}) \circledcirc \mathcal{B})\ .$$

Le produit coloré sert à choisir les atomes appartenant à la première part de la partition (ceux colorier avec la première couleur) et ceux appartenant à la sous-partition.

On obtient donc une équation fonctionnelle vérifiée par la série génératrice ordinaire des nombres de Bell :

$$B(z) = 1 + z \cdot \mathcal{L}_{\mathrm{c}} \left(\mathcal{B}_{\mathrm{c}}(B(z)) \cdot \mathcal{B}_{\mathrm{c}} \left(\frac{1}{1-z} \right) \right).$$

En appliquant la transformation de Borel, et en utilisant une des identités données dans l'annexe B, on obtient :

$$\mathcal{B}_{\mathrm{c}}(B(z))' = \mathcal{B}_{\mathrm{c}}(B(z))\, e^{z}.$$

On résout simplement cette équation différentielle et on obtient le résultat attendu :

$$\mathcal{B}_{\mathrm{c}}(B(z)) = e^{e^{z}-1}.$$

CHAPITRE 3

Génération aléatoire

Ce chapitre traite de la génération aléatoire uniforme des différentes structures précédemment présentées dans ce manuscrit : les diamants et les étiquetages croissants de graphes Fork-Join.

Dans le cas des diamants, la génération aléatoire uniforme permet d'observer empiriquement les résultats obtenus théoriquement. On peut aussi l'utiliser pour générer des programmes aléatoires, dans le cadre du test de compilateurs ou d'outils d'analyse statique [YCER11]. De plus, le cadre théorique de la génération de Boltzmann [DFLS04], dans lequel nous nous plaçons, permet une grande généricité et modularité. Donc, ces algorithmes peuvent être utilisés plus généralement pour la génération aléatoire de structures croissantes définies à l'aide du produit boîte.

Comme pour les diamants, la génération aléatoire d'extensions linéaires a pour motivations ses vastes applications dans différents domaines : dans la vérification (*model-checking* statistique, cf. [GS05]) et le test aléatoire (cf. [Sen07]) de programmes concurrents, les problèmes de classement [SI09] ou d'inférence bayésienne [EW08] et en bio-informatique [MS15]. Ici, nous présenterons un algorithme de génération aléatoire uniforme d'extensions linéaires pour les ordres partiels Série-Parallèle, ou dit autrement, un algorithme de génération d'étiquetages croissants de graphes Fork-Join.

Dans la suite de ce chapitre, nous présenterons les algorithmes de Boltzmann pour la génération aléatoire de diamants croissants ainsi que quelques expérimentations. Puis nous ferons de même pour l'algorithme de génération aléatoire d'étiquetages croissants de graphes Fork-Join.

3.1 Génération aléatoire uniforme de Boltzmann

3.1.1 Introduction

Dans la suite de cette section, nous rappellerons les grandes lignes de la méthode de Boltzmann introduite par [DFLS04]. Puis en nous basant sur les travaux de O. Bodini [Bod10] et [BRS12] (avec O. Roussel et M.soria), nous proposerons de nouveaux générateurs de Boltzmann pour les structures étiquetées croissantes.

La génération de Boltzmann de structures étiquetées génère uniquement les structures sans leurs étiquettes. Les diamants étant des graphes Fork-Join, on pourra donc naturellement utiliser l'algorithme de génération d'étiquetage présenté dans la section 3.3.

3.1.2 Distribution de Boltzmann pour les classes étiquetées

La méthode de Boltzmann consiste à construire automatiquement un générateur aléatoire d'une classe de structures $\mathcal{F}$. Pour garantir une complexité linéaire du générateur, la génération se fait en taille approchée : dans un intervalle $[n-\epsilon, n+\epsilon]$ où n est la taille visée.

La loi de probabilité des structure générées est une loi dite de Boltzmann de paramètre x, c'est-à-dire, pour un objet γ de $\mathcal{F}$:

$$\mathbb{P}_x(\gamma) = \frac{x^{|\gamma|}}{|\gamma|!\, F(x)} \qquad \text{avec } F(x) = \sum_{n\geqslant 0} f_n \frac{z^n}{n!}.$$

Remarquons que chaque objet de $\mathcal{F}$, de taille n, a la même probabilité d'être tiré. La génération aléatoire de Boltzmann est donc uniforme pour les objets de même taille :

$$\begin{aligned}\mathbb{P}_x(N=n) &= \sum_{\gamma\in\mathcal{F}\wedge|\gamma|=n} \mathbb{P}_x(\gamma)\\ &= f_n \frac{x^n}{n!\, F(x)}.\end{aligned}$$

Il est alors direct de calculer l'espérance de N :

$$\begin{aligned}\mathbb{E}_x(N) &= \sum_{n\geqslant 0} n\mathbb{P}_x(N=n)\\ &= \frac{x}{F(x)} \sum_{n\geqslant 0} \cdot f_n \cdot \frac{x^{n-1}}{(n-1)!}\\ &= x\frac{F'(x)}{F(x)}.\end{aligned}$$

De cette équation on peut calculer x de façon à obtenir l'espérance souhaitée pour la taille des objets à générer. Dans le cas où l'on veut générer de très grosses structures, on peut aussi choisir de prendre un point x proche de la singularité dominante ρ de F pour obtenir une espérance infinie et donc générer de grandes structures.

3.1.3 Générateurs de Boltzmann usuels

Maintenant que nous avons décrit la sortie d'un générateur de Boltzmann, décrivons sa construction. Comme nous l'avons dit, un des intérêts principaux de la méthode de Boltzmann est de construire automatiquement un algorithme de génération aléatoire uniforme pour des objets d'une classe $\mathcal{F}$ à partir de sa spécification. On note $\Gamma[\mathcal{F}](x)$ le générateur de Boltzmann de paramètre x, d'une classe étiquetée $\mathcal{F}$.

Commençons par le cas simple où $\mathcal{F} = \mathcal{A} + \mathcal{B}$:
On veut construire un générateur tel qu'un objet γ de $\mathcal{F}$,soit générer avec probabilité $\frac{x^{|\gamma|}}{|\gamma|!\ F(x)} = \frac{x^{|\gamma|}}{|\gamma|!\ (A(x)+B(x))}$. Remarquons que la probabilité que γ soit dans $\mathcal{A}$ est

$$\begin{aligned}
\mathbb{P}_x(\gamma \in \mathcal{A} | \gamma \in \mathcal{F}) &= \sum_{n \geqslant 0} \sum_{\gamma \in \mathcal{F} \wedge |\gamma| = n} \frac{x^n}{n!\ (A(x) + B(x))} \\
&= \sum_{n \geqslant 0} a_n \frac{x^n}{n!\ (A(x) + B(x))} \\
&= \frac{A(x)}{A(x) + B(x)}.
\end{aligned}$$

De la même façon, on calcule $\mathbb{P}_x(\gamma \in \mathcal{B} | \gamma \in \mathcal{F}) = \frac{B(x)}{A(x)+B(x)}$.

En supposant que nous disposions de générateur de Boltzmann pour $\mathcal{A}$ et $\mathcal{B}$, on en déduit donc l'algorithme 1 qui construit un générateur $\Gamma[\mathcal{F}](x)$ à partir de générateurs $\Gamma[\mathcal{A}](x)$ et $\Gamma[\mathcal{B}](x)$.

Algorithme 1 Générateur de Boltzmann de $\mathcal{F} = \mathcal{A} + \mathcal{B}$ et de paramètre x

```
function Γ[F](x)
    if Bernoulli(A(X)/(A(x)+B(x))) then
        return Γ[A](x)
    else
        return Γ[B](x)
```

Dans le cas du produit, c'est-à-dire $\mathcal{F} = \mathcal{A} \star \mathcal{B}$, l'algorithme est encore plus simple. Il suffit d'observer que

$$\begin{aligned}
\mathbb{P}_x((\alpha, \beta)) &= \frac{x^{|\alpha|+|\beta|}}{(|\alpha| + |\beta|)!\ F(x)} \\
&= \frac{x^{|\alpha|+|\beta|}}{(|\alpha| + |\beta|)!\ (A(x) \cdot B(x))} \\
&= \frac{1}{\binom{|\alpha|+|\beta|}{|\alpha|}} \frac{x^{|\alpha|}}{|\alpha|!\ A(x)} \frac{x^{|\beta|}}{|\beta|!\ B(x)}.
\end{aligned}$$

Le coefficient binomial $\binom{|\alpha|+|\beta|}{|\alpha|}$ étant la conséquence du fait que nous travaillons avec des objets étiquetés. On peut donc directement dériver l'algorithme 2 de cette égalité.

Algorithme 2 Générateur de Boltzmann de $\mathcal{F} = \mathcal{A} \star \mathcal{B}$ et de paramètre x

function $\Gamma[\mathcal{F}](x)$
  return $(\Gamma[\mathcal{A}](x), \Gamma[\mathcal{B}](x))$

De même, on peut dériver les algorithmes pour les opérateurs standards SEQ, SET et CYC. On trouvera le détails de ces algorithmes dans l'article original [DFLS04].

3.1.4 Générateurs de Boltzmann pour les équations différentielles du premier ordre

On rappelle que l'un de nos objectifs initiaux est d'obtenir un générateur de Boltzmann pour les diamants croissants du chapitre 1. Or les séries génératrices de ces objets sont solutions d'équations différentielles du deuxième ordre. Mais, le principe des générateurs de Boltzmann pour les équations du deuxième ordre est juste une adaptation de celui pour les équations du premier ordre. C'est pour cela que nous détaillons les précédents travaux de O. Bodini *et al.*.

Cas général

On s'intéresse ici à la construction de générateurs de Boltzmann pour les classes spécifiées à l'aide d'un produit boîte. Plus exactement, on considère les classes dont la série génératrice F est solution d'une équation différentielle du premier ordre

$$\begin{aligned} & F'(z) = \phi(F(z), z) \qquad \text{et } F(0) \text{ quelconque} \\ \text{ou} \quad & F(z) = \int_0^t \phi(F(t), t)\mathrm{d}t + F(0). \end{aligned}$$

L'idée principale qui permet de construire de tels générateurs de Boltzmann est de modifier le paramètre x au fur et à mesure de la génération aléatoire. Cette modification se fait par multiplication de x par une variable aléatoire U distribuée selon une loi de densité

$$\delta_x^F(u) = \frac{xF'(ux)}{F(x) - F(0)} = \frac{x\phi(F(ux), ux)}{F(x) - F(0)}$$

L'algorithme 3 donne le pseudo-code nécessaire à la construction d'un générateur de Boltzmann pour $\mathcal{F}$ à partir de $\mathcal{F}' = \phi(\mathcal{F}, \mathcal{Z})$. Dans ce cas, on assimilera l'opérateur de dérivation à la suppression de l'atome de plus grande étiquette.

On constate que l'algorithme 3 construit bien un objet de $\mathcal{F}$: soit il retourne un objet de taille nulle, soit il retourne un objet généré de la classe $\phi(\mathcal{F}, \mathcal{Z})$ c'est-à-dire un objet de $\mathcal{F}'$, autrement dit, un objet de $\mathcal{F}$ privé de son atome de plus grande étiquette.

Algorithme 3 Générateur de Boltzmann de $\mathcal{F}$ à partir de $\mathcal{F}' = \phi(\mathcal{F}, \mathcal{Z})$ et de paramètre x

function $\Gamma[\mathcal{F}](x)$
  if BERNOULLI$\left(\frac{F(0)}{F(x)}\right)$ **then**
    return un objet de $\mathcal{F}$ de taille 0, choisi uniformément
  else
    Tirer U dans $[0, 1]$ selon la densité δ_x^F
    Générer un objet γ' à partir de $\Gamma[\phi(\mathcal{F}, \mathcal{Z})](Ux)$
    return l'objet $(\mathcal{Z}, \gamma')$ où $\mathcal{Z}$ a la plus grande étiquette

Pour comprendre pourquoi cet objet est généré selon la bonne distribution (celle de Boltzmann), il suffit de calculer la probabilité de tirer un objet γ de taille $n + 1$.

$$\begin{aligned}
\mathbb{P}(\Gamma[\mathcal{F}](x) = \gamma) &= \left(1 - \frac{F(0)}{F(x)}\right) \int_0^1 \mathbb{P}(\Gamma[\phi(\mathcal{F}, \mathcal{Z})](ux) = \gamma') \cdot \delta_x^F(u)\, \mathrm{d}u \\
&= \frac{F(x) - F(0)}{F(x)} \int_0^1 \frac{(ux)^n}{n!\, \phi(F(ux), ux)} \frac{x\phi(F(ux), ux)}{F(x)_F(0)}\, \mathrm{d}u \\
&= \frac{x^{n+1}}{(n+1)!\, F(x)}
\end{aligned}$$

Pour une preuve plus complète de la correction de l'algorithme 3, on peut se référer à [BRS12].
Pour compléter la description de cet algorithme il reste à décrire comment tirer une variable aléatoire de densité δ_x^F et comment évaluer F.

Pour l'évaluation, on peut montrer que le choix, simple, d'utiliser un développement (borné) de Taylor en 0 de F est suffisamment précis pour garantir l'efficacité de l'algorithme. Mais de fait, cela introduit un biais sur la distribution des tailles des objets qui sont donc généralement plus petits.

Pour le tirage de U, on peut utiliser un algorithme tel que l'algorithme du Ziggourat (cf. [MT00]). On peut aussi raisonner simplement sur la fonction de répartition de δ_x^F. On a

$$\mathbb{P}_x(U \leqslant t) = \int_0^t \frac{xF'(ux)}{F(x) - F(0)}\, \mathrm{d}u = \frac{F(tx) - F(0)}{F(x) - F(0)}.$$

On peut donc tirer une variable aléatoire V uniforme sur $[0, 1]$ et calculer numériquement pour quelle valeur de t on a $V = \mathbb{P}_x(U \leqslant t)$: on doit résoudre, en t, l'équation suivante

$$F(tx) = VF(x) + (1 - V)F(0). \tag{3.1.1}$$

Étant donné la croissance de δ_x^F (Lemme 2.2 de [BRS12]), on peut utiliser une recherche dichotomique ou une itération numérique, par exemple une itération de Newton numérique, pour résoudre cette équation (3.1.1).

Cas autonome

Dans le cas où l'équation différentielle considérée est autonome, c'est-à-dire quand $F' = \phi(F)$, l'algorithme peut être largement simplifié. Dans ce cas, la valeur de Ux n'est pas nécessaire, seule la valeur de $F(Ux)$ l'est. On peut donc simuler la variable de densité δ_x^F directement en utilisant la relation (3.1.1).

Dans ce cas, l'algorithme n'a donc pas besoin de simuler autre chose que des variables aléatoires de loi uniforme pour calculer $F(ux)$. On peut notamment, en posant $\tau = F(x)$, voir l'algorithme non plus comme un générateur de Boltzmann de paramètre x mais comme un générateur de Boltzmann de paramètre τ. Ces modifications sont détaillées dans l'algorithme 4.

Algorithme 4 Générateur de Boltzmann de $\mathcal{F}$ à partir de $\mathcal{F}' = \phi(\mathcal{F})$

function $\Gamma[\mathcal{F}](\tau)$
  if $\textsc{Bernoulli}\left(\frac{F(0)}{\tau}\right)$ **then**
    return un objet de $\mathcal{F}$ de taille 0, choisi uniformément
  else
    Tirer U de loi uniforme sur $[0, 1]$
    $\tau \leftarrow U\tau + (1 - U)F(0)$
    Générer un objet γ' à partir de $\Gamma[\phi(\mathcal{F})](\tau)$
    return l'objet $(\mathcal{Z}, \gamma')$ où $\mathcal{Z}$ a la plus grande étiquette

3.1.5 Générateurs de Boltzmann pour les équations différentielles du second ordre

Nous pouvons maintenant présenter nos générateurs de Boltzmann pour les équations différentielles du second ordre. Comme pour les équations différentielles du premier ordre, le cas autonome permet des simplifications.

Cas général

Notre problème consiste en la construction d'un générateur de Boltzmann pour des objets dont la série génératrice vérifie à une équation différentielle du second ordre $F''(z) = \phi(F(z), z)$.

Une première idée est d'utiliser deux fois l'algorithme 3 : un première fois pour générer un γ' à partir de $\Gamma_x[\mathcal{F}'']$, puis une seconde fois pour générer un γ'' à partir de ce générateur de $\Gamma_x[\mathcal{F}']$.

Si l'on utilise ce principe, on constate qu'on a alors besoin du produit de deux variables U et V distribuées selon deux distributions δ^F et $\delta^{F'}$. Une simplification est donc possible en ne générant qu'une seule variable $S = UV$. Pour trouver la distribution de S, il suffit de résoudre

en $\hat{\delta}_x^F$ l'équation suivante, où n désigne la taille de γ

$$\begin{aligned}\mathbb{P}(\Gamma[\mathcal{F}](x) = \gamma) = \frac{x^n}{n!\,F(x)} &= (1 - \frac{F(0) + xF'(0)}{F(x)}) \int_0^1 \mathbb{P}\left(\Gamma[\phi(\mathcal{F}'', \mathcal{Z})](sx) = \gamma''\right) \hat{\delta}_x^F(s)\mathrm{d}s \\ &= \frac{F(x) - F(0) - xF'(0)}{F(x)} \int_0^1 \frac{(sx)^{n-2}}{(n-2)!\,F''(sx)} \hat{\delta}_x^F(s)\mathrm{d}s \\ &= \frac{F(x) - F(0) - xF'(0)}{F(x)} \frac{x^{n-2}}{(n-2)!} \int_0^1 \frac{s^{n-2}}{F''(sx)} \hat{\delta}_x^F(s)\mathrm{d}s\end{aligned}$$

On trouve alors la solution

$$\hat{\delta}_x^F(s) = \frac{x^2(1-s)F''(sx)}{F(x) - F(0) - xF'(0)}.$$

On en déduit donc l'algorithme 5, un générateur de Boltzmann $\Gamma[\mathcal{F}]$ à partir de $\Gamma[\mathcal{F}'']$. On génère un objet $\gamma'' \in \mathcal{F}''$ auquel on ajoute un atome de plus petite étiquette et un de plus grande.

Algorithme 5 Générateur de Boltzmann de $\mathcal{F}$ à partir de $\mathcal{F}'' = \phi(\mathcal{F}, \mathcal{Z})$

function $\Gamma[\mathcal{F}]$(x)
  $W \sim \mathscr{U}([0, 1])$ ▷ $\mathscr{U}$ désigne la loi uniforme
  if $W < \left(\frac{F(0)}{F(x)}\right)$ **then**
    return un objet de $\mathcal{F}$ de taille 0, choisi uniformément
  else if $W < \left(\frac{F(0)+xF'(0)}{F(x)}\right)$ **then**
    return un objet de $\mathcal{F}$ de taille 1, choisi uniformément
  else
    Tirer S dans $[0, 1]$ selon la densité $\hat{\delta}_x^F$
    Générer un objet γ'' à partir de $\Gamma[\phi(\mathcal{F}, \mathcal{Z})](Sx)$
    return l'objet $(\mathcal{Z}_1, \gamma'', \mathcal{Z}_2)$ où $\mathcal{Z}_1$ a la plus grande étiquette et $\mathcal{Z}_2$ la plus petite

Cas autonome

Comme dans le cas des équations différentielles de premier ordre, le cas autonome $F''(z) = \phi(F(z))$ induit des simplifications pour l'algorithme.

La première idée est de réutiliser le principe d'inversion de la fonction de répartition :

$$\mathbb{P}(S \leqslant t) = \int_0^t \frac{x^2(1-s)F''(sx)}{F(x) - F(0) - xF'(0)}\mathrm{d}s.$$

Or, comme nous le constatons, il semble difficile de calculer cette intégrale dans le cas général.

Une autre possibilité est d'appliquer deux fois l'algorithme 4. Pour ce faire, on essaye de se ramener à une équation du type $F' = \Psi(F)$.

En multipliant $F''(z) = \phi(F(z))$ par $2F'$ puis en intégrant les deux côtés, on obtient

$$F'(z) = \sqrt{2\Phi(F(z)) - 2\Phi(F(0)) + F'(0)} \tag{3.1.2}$$

où $\Phi(t) = \int_0^t \phi(u)\mathrm{d}u$.

Maintenant, il ne reste plus qu'à inverser cette équation pour obtenir le bon paramètre pour l'appel du générateur $\Gamma[\phi(\mathcal{F})]$. On rappelle que l'algorithme 4 fournit un générateur $\Gamma[F]$ à partir de $\Gamma[F']$. On cherche maintenant à obtenir un générateur de $\Gamma[F'']$ à partir d'un générateur $\Gamma[F']$.

Toujours d'après le raisonnement sur l'algorithme 4, pour obtenir le générateur $\Gamma[F'']$ on multiplie le paramètre y par une variable aléatoire que la relation suivante nous abstient de simuler : $F'(ty) = VF'(y) + (1 - V)F'(0)$ avec V une variable aléatoire de loi uniforme sur $[0, 1]$.
Or, le paramètre y est lui même de la forme $F(x)$ car c'est celui d'un générateur $\Gamma[\mathcal{F}'](\tau)$ obtenu à l'aide de l'algorithme 4. On cherche maintenant une relation sur F permettant de mettre à jour y.

Après inversion de l'équation (3.1.2), on obtient

$$\begin{aligned} F(z) &= \Phi^{-1}(\frac{1}{2}(F'^2(z) + 2\Phi(F(0)) - F'(0)^2)) \\ &= \Psi(F'(z)) \end{aligned}$$

où Φ^{-1} désigne l'inverse compositionnel de Φ. En utilisant la relation (3.1.1), on obtient ainsi une nouvelle relation

$$F(t\tau) = \Psi\left(V\Psi^{-1}(F(\tau)) + (1 - V)F'(0)\right).$$

On peut alors utiliser cette relation et la précédente pour mettre à jour le paramètre de Boltzmann dans l'algorithme 6.

3.1.6 Complexité

Comme nous l'avons vu dans les sections précédentes les objets pour lesquels nous utilisons les générateurs de Boltzmann présentés ont tous une singularité dominante de type pôle. Dans ces cas-là, le résultat de [DFLS04] nous garantit une complexité linéaire en la taille des objets générés pour la génération en taille approchée.

3.2 Expérimentations : les diamants généraux

Pour terminer ce chapitre, nous présentons différentes expériences basées sur la génération aléatoire de diamants plans croissants ou non. Nous considérerons deux modèles : le modèle de diamants généraux croissants utilisant l'algorithme présenté dans la section précédente, et le modèle de diamants généraux non étiquetés $\mathcal{N} = \mathcal{Z} + \mathcal{Z} \times \textsc{Seq}(\mathcal{N}) \times \mathcal{Z}$ pour lesquels on tire aléatoirement et uniformément un étiquetage croissant.

Algorithme 6 Générateur de Boltzmann de $\mathcal{F}$ à partir de $\mathcal{F}'' = \phi(\mathcal{F})$ avec $\tau = F(x)$

function $\Gamma[\mathcal{F}](\tau)$
  if $\textsc{Bernoulli}\left(\frac{F(0)}{\tau}\right)$ **then**
    return un objet de $\mathcal{F}$ de taille 0, choisi uniformément
  else
    $U \sim \mathscr{U}([0, 1])$
    $\sigma \leftarrow U\tau + (1 - U)F(0)$ ▷ mise à jour pour $\Gamma[F']$
    if $\textsc{Bernoulli}\left(\frac{F'(0)}{\Psi^{-1}(\sigma)}\right)$ **then**
      return un objet de $\mathcal{F}$ de taille 1, choisi uniformément
    else
      $V \sim \mathscr{U}([0, 1])$
      $\tau \leftarrow \Psi(V\Psi^{-1}(\sigma) + (1 - V)F'(0))$ ▷ mise à jour pour $\Gamma[F'']$
      Générer un objet γ'' à partir de $\Gamma[\phi(\mathcal{F}, \mathcal{Z})](\tau)$
      return l'objet $(\mathcal{Z}_1, \gamma'', \mathcal{Z}_2)$ où $\mathcal{Z}_1$ a la plus grande étiquette et $\mathcal{Z}_2$ la plus petite

Ensuite, en utilisant la bijection entre diamants plans croissants et graphes cactus croissants, nous mènerons une étude empirique permettant de donner quelques intuitions sur les paramètres moyens de ces différents types de structures.

Nous avons choisi d'exploiter la bijection avec les graphes cactus car ceux-ci semblent les moins étudiés dans la littérature contrairement aux deux autres familles d'arbres croissants, étudiées dans de nombreux articles (par exemple [BFS92]).

3.2.1 Algorithmes de génération aléatoire uniforme

Diamants plans croissants

Ce modèle de diamants correspond à la spécification 1.3.1 page 17 :

$$\mathcal{F} = \mathcal{Z} + \mathcal{Z}^{\square} \star \textsc{Seq}\,(\mathcal{F}) \star \mathcal{Z}^{\blacksquare}.$$

Un générateur de Boltzmann pour ces diamants plans croissants est aisément obtenu en combinant l'algorithme 6 avec un générateur de Boltzmann pour l'opérateur $\textsc{Seq}$. Dans ce cas, Ψ et Ψ^{-1} ont des formes très simples :

$$\Psi(x) = 1 - \frac{2}{x^2 + 1}, \qquad \Psi^{-1}(x) = \sqrt{\frac{2}{1 - x} - 1}.$$

Le générateur pour la séquence est celui proposé dans l'article original [DFLS04] :

Algorithme 7 Générateur de Boltzmann de $\textsc{Seq}(\mathcal{F})$ à partir de celui de $\mathcal{F}$

function $\Gamma[\textsc{Seq}(\mathcal{F})](x)$
  $k \leftarrow \textsc{Geom}(F(x))$ $\triangleright\ \mathbb{P}(\textsc{Geom}(p) = k) = p^{k-1}(1-p)$
  return le n-uplet $(\Gamma[\mathcal{F}](x), \ldots, \Gamma[\mathcal{F}](x))$ de taille k

où $\textsc{Geom}$ désigne la loi géométrique.

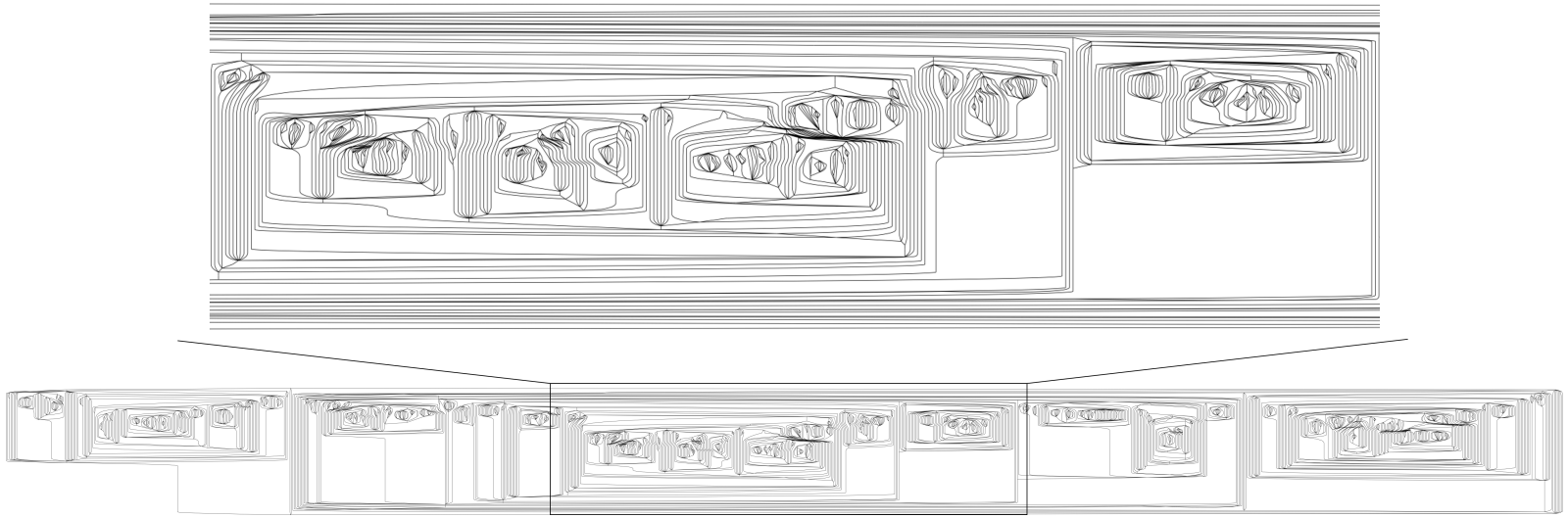


FIGURE 3.1 – Diamant plan croissant de taille 3423

On remarque que pour implémenter ce générateur pour $\textsc{Seq}(\mathcal{F})$ il est nécessaire d'évaluer F en différents points. Dans notre cas F est définie implicitement comme étant l'inverse compositionnel de la fonction d'erreur erf [1]. Cette définition n'étant pas des plus simples à exploiter pour obtenir des évaluations numériques de F nous avons fait le choix d'approximer F par son développement de Taylor en 0 à l'ordre 300. Ce choix empirique, a été suffisamment précis pour nous permettre de générer des structures de plusieurs milliers de noeuds comme celle présentée figure 3.1. Enfin, nous avons de choisi comme paramètre de Boltzmann initial, une approximation fine de ρ (nécessitant les 64 bits d'un flottant machine), calculée à l'aide de la formule 1.4.3 page 26 :

$$\rho \simeq 0.6556795424187984.$$

L'étiquetage croissant des diamants est tiré uniformément parmi tous les étiquetages croissants possibles à l'aide de l'algorithme 9, présenté dans la section suivante.

Diamants plans ordinaires

Ce modèle de diamant non étiqueté correspond à la spécification 1.3.1 page 17 :

$$\mathcal{F} = \mathcal{Z} + \mathcal{Z} \times \textsc{Seq}\,(\mathcal{F}) \times \mathcal{Z}.$$

Cette spécification est très classique au sens où on peut la voir comme une spécification d'arbres généraux dont les noeuds internes seraient comptés deux fois et les feuilles une seule fois. Un rapide calcul nous donne une formule pour F :

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

De cette formule, on trouve directement que la singularité dominante provient de l'annulation du radical, et donc on calcule la plus petite racine réelle positive de $1 - 2z - 3z^2$ qui est $\frac{1}{3}$. On remarque que

$$F(\frac{1}{3}) = \frac{1}{3} + \frac{\left(\frac{1}{3}\right)^2}{1 - \frac{2}{3}} = \frac{2}{3}.$$

On peut donc facilement construire le générateur de Boltzmann singulier (algorithme 8) à partir de l'algorithme 7.

De même que pour le cas croissant, on utilisera l'algorithme du chapitre suivant pour la génération de l'étiquetage croissant.

3.2.2 Comparaison des deux modèles de diamants

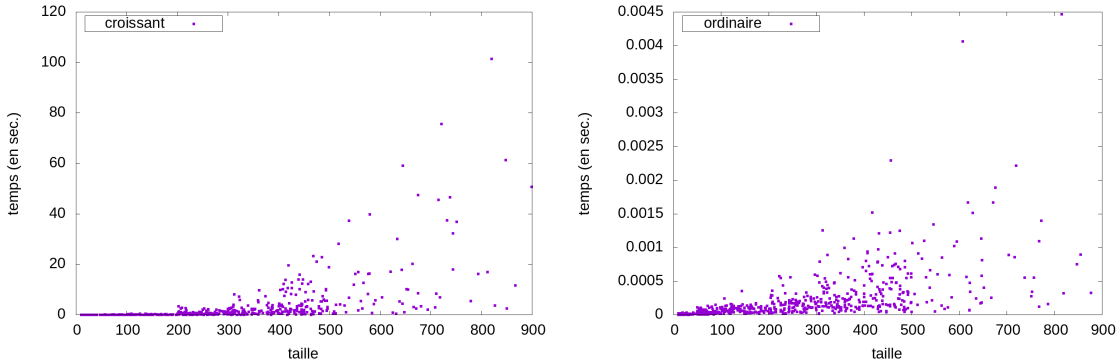
La première comparaison que l'on peut faire entre ces deux modèles est la comparaison des performances de leur algorithme de génération de Boltzmann associé.

Comme on le constate sur la figure 3.2, le temps pour générer des diamants ordinaires (non étiquetés) est très rapide : de l'ordre de quelques centièmes de secondes pour des diamants de

1. $\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2}\, \mathrm{d}t.$

Algorithme 8 Générateur de Boltzmann de diamants plans non étiquetés

function $\Gamma[\mathcal{F}]$
  if $\textsc{Bernoulli}(\frac{1}{2})$ **then**
    return un atome $\bullet$
  else
    $k \leftarrow \textsc{Geom}(\frac{2}{3})$
    return $(\bullet, \Gamma[\mathcal{F}](), \ldots, \Gamma[\mathcal{F}](), \bullet)$ avec k appels à $\Gamma[\mathcal{F}]$



(a) Diamants croissants (b) Diamants ordinaires

FIGURE 3.2 – Temps de génération en fonction de la taille

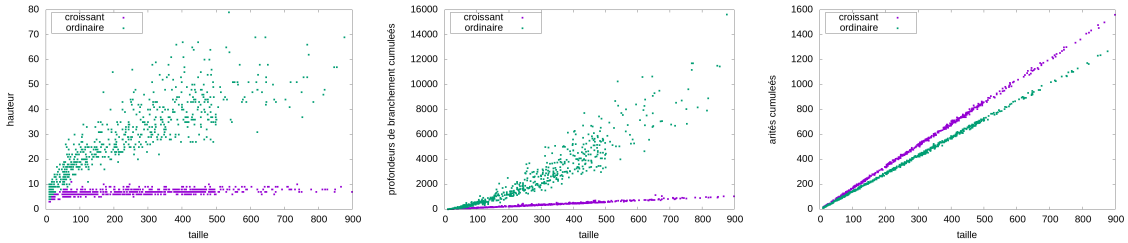
taille inférieure à 900. Dans le cas étiqueté, ce temps peut considérablement s'allonger et atteindre une minute, même si il est généralement en dessous de 20 secondes. Néanmoins, on est capable de générer des diamants croissants de plusieurs milliers de noeuds : par exemple 4000 noeuds en 200 secondes. Pour les diamants ordinaires, on génère des objets de quelques millions de noeuds en moins d'une seconde.

Ces grosses différences d'efficacité s'expliquent par le coût et la perte de précisions dûs à l'évaluation de F à quasiment chaque itération dans le cas des diamants croissants. Dans le cas ordinaire, aucun calcul n'est nécessaire et la précision est maximale.

Maintenant cette étude faite, on se concentre sur les différences de comportement structurel de ces deux modèles de diamants plans. Par la suite, nous considérerons les trois paramètres suivants :

— la hauteur : c'est la longueur du plus long chemin de la source au puits d'un diamant,
— l'arité cumulée : c'est la somme des degrés sortants de chaque noeud du diamant,
— la profondeur de branchement cumulée : pour chaque noeud, on définit la profondeur de branchement comme étant son nombre d'ancêtres "branchants" (ancêtre de la séquence dans la spécification).

Dans le cas non étiqueté, les lois de probabilité limites pour la hauteur (cf. [dBKR71]) et l'arité cumulée (obtenue simplement depuis la série génératrice) sont bien connues, on en trouve différentes analyses dans la littérature. La profondeur de branchement cumulée peut



(a) Hauteur (b) Profondeurs de branchement cumulées (c) Arités cumulées

Les mesures correspondantes au modèle croissant sont en violet (points plus foncés) et celles du modèle ordinaire en vert.

FIGURE 3.3 – Différents paramètres sur les diamants

être réinterprétée comme la longueur de cheminement, dont la loi de probabilité limite est aussi connue (cf. [Tak91]) et dont on peut aussi obtenir les moments par analyse de la série génératrice.

Pour le cas croissant, on s'attend à ce que la hauteur soit assez petite (devant celle des diamants ordinaires) et donc la profondeur de branchement aussi. Pour l'arité cumulée, on s'attend à ce qu'elle soit plus élevée. On suppose aussi que les diamants croissants sont plus aplatis et contiennent moins de chaînes d'atomes, au vu de l'arité moyenne de la racine (cf. section 1.5.5).

Pour confirmer cette intuition, nous avons généré uniformément des diamants croissants et ordinaires, de taille 10 à 900 et calculer ces différentes statistiques sur chacun. Les résultats sont récapitulés dans la figure 3.3.

Comme prévu, nos intuitions sont confirmées par ces expériences. On exploite maintenant la bijection de la section 1.3.1 pour faire quelque expérimentations sur le modèle de graphes cactus.

3.2.3 Comparaison entre les modèles de cactus

La bijection entre diamants plans et cactus est facilement implémentée. Dans la figure 3.4, on peut observer un exemple de cactus obtenu en appliquant la bijection au diamant figure 3.1. Les arêtes rouges sont celles appartenant à des cycles et le noeud bleu correspond au noeud d'étiquette 0.

Après plusieurs simulations, on décide de regarder deux paramètres qui semblent d'intérêt sur les cactus : le nombre de cycles et le diamètre du graphe (longueur du plus long chemin de la source à un puits).

Pour ces deux paramètres, on regarde les histogrammes normalisés (on a retranché la moyenne et divisé par la racine carré de la variance) des paramètres de diamants face à ceux des cactus. La figure 3.5 présente ces histogrammes pour des diamants de taille 100. C'est la taille la plus élevée pour laquelle nous avons réussi à générer un grand nombre de diamants (environ 50000) en un temps raisonnable (quelques heures).

Même si ces histogrammes n'ont qu'une valeur empirique, ils nous permettent d'émettre quelques hypothèses qu'il serait intéressant de vérifier théoriquement dans de futurs travaux :

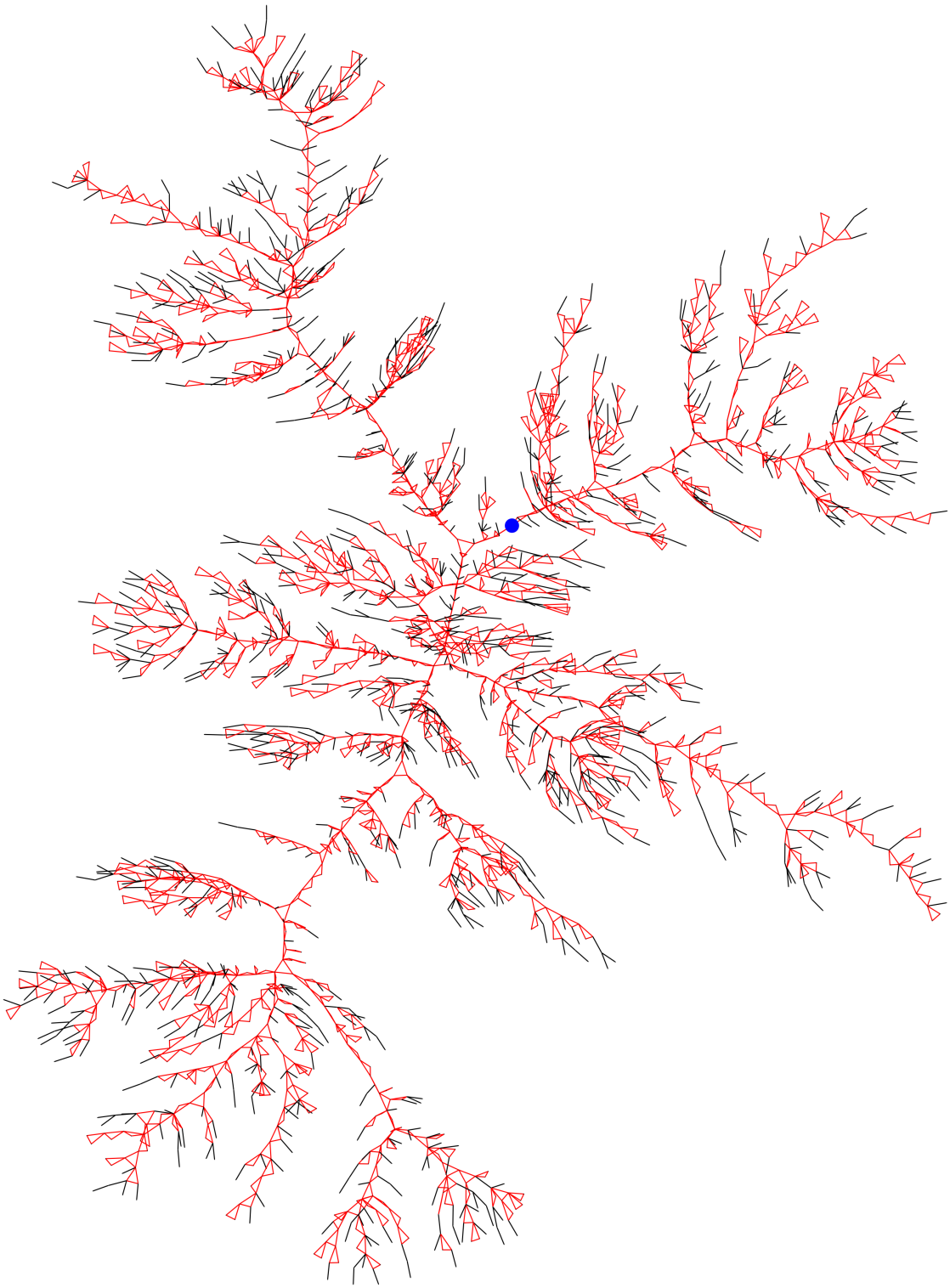
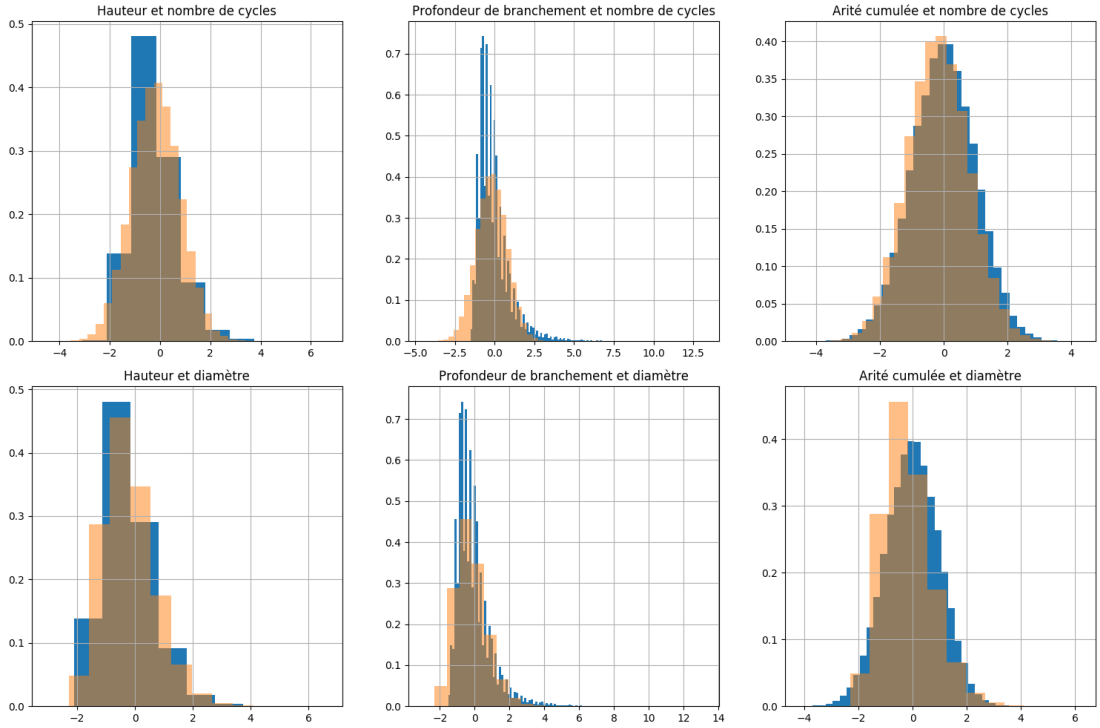
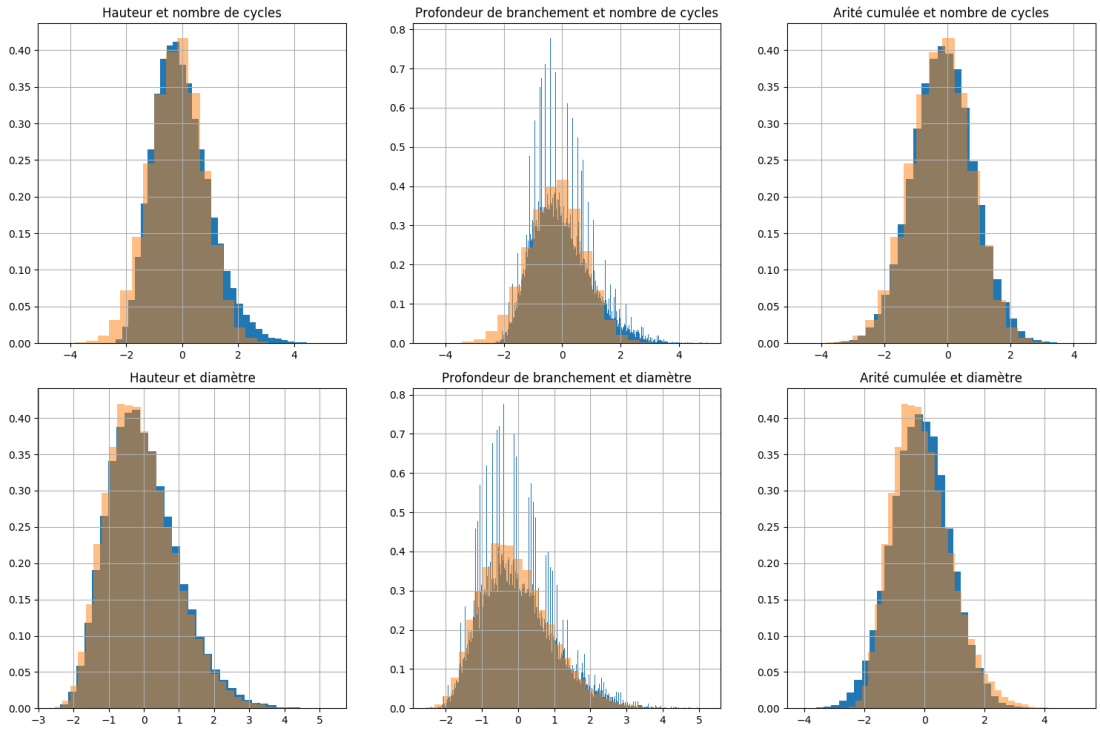


FIGURE 3.4 – Cactus croissant de taille 3422



(a) Diamants croissants



(b) Diamants ordinaire

Les histogrammes en bleu correspondent aux paramètres des diamants et ceux en orange à ceux des cactus.

FIGURE 3.5 – Histogrammes de différents paramètres sur des diamants de taille 100

— dans les deux modèles (croissant et ordinaire), les distributions de la profondeur de branchement cumulée dans les diamants et du diamètre dans les cactus semblent fortement corrélées,
— il en est de même pour les lois de l'arité cumulée des diamants et du nombre de cycles des cactus

On observe aussi une corrélation entre hauteur et diamètre, mais vu la différence de comportement entre le modèle croissant et le modèle ordinaire, il semblerait que le phénomène soit plus probablement une expression de la corrélation "évidente" entre hauteur et profondeur de branchement dans les diamants.

3.3 Extensions linéaires d'ordres Série-Parallèle

3.3.1 Introduction

Nous présentons maintenant un algorithme de génération aléatoire uniforme d'extensions linéaires d'ordres Série-Parallèle. Un point a souligné est le caractère optimal de cet algorithme en terme de nombre de bits aléatoires utilisés.

Dans la suite de cette section, nous présentons des travaux connexes à cette problématique, puis nous décrirons l'algorithme et présenterons quelques expérimentations. L'algorithme utilise un certain nombre de *primitives stochastiques* que nous détaillerons dans une sous-section dédiée, du fait de leur généricité.

3.3.2 Travaux connexes

Il y a peu de travaux existants à propos de la génération aléatoire d'extensions linéaires.

La plupart utilise des techniques probabilistes du type *Monte-Carlo* [BD99] (approximation) ou *Couplage arrière* (exacte) [Hub06] (*Coupling from the past* en anglais). Tous ces travaux se basent une chaîne de Markov dites de Karzanov et Kachiyan [KK91]. Un intérêt différent du nôtre est commun à ces différents travaux : la mise au point de *schéma d'approximation randomisé entièrement en temps polynomiale* [JVV86] (*Fully Polynomial Randomized Approximation Scheme (FPRAS)* dans la littérature) pour le calcul de volume de corps convexes. On reviendra sur ce dernier point dans le chapitre suivant.

Dans un autre registre, on peut aussi citer les travaux consistants à énumérer toutes les extensions linéaires ([DLDMDB06], [PR94]) . On peut utiliser ces algorithmes pour générer toutes les extensions linéaires puis en choisir une au hasard.

Enfin, il y a le précédent travail de mes co-auteurs qui traitent de ce problème dans le cas des *tree posets* [BGP16].

3.3.3 Algorithmes

La structure de données sur laquelle travaille l'algorithme est celle des graphes Fork-Join bicolores présentés dans le chapitre précédent (définition 5). On rappelle que seuls les noeuds noirs font partie de l'extension linéaire à générer, contrairement aux noeuds blancs qui son "muets".

Cet algorithme peut-être décrit de deux manières totalement équivalentes qui ont chacune leur intérêt. Pour mieux comprendre la suite, l'intuition est de voir les graphes Fork-Join comme des arbres où le graphe du dessous est recollé à la racine comme fils le plus à droite (pour le distinguer). En fait, ce recollage définit une bijection entre graphes Fork-Join et une certaines familles d'arbres unaires-ternaires.

Par ce prisme, on peut définir l'algorithme par un schéma algorithmique *bottom-up* ou *top-down*. L'approche *bottom-up* a l'intérêt d'être simple à décrire et facilite le raisonnement car elle reprend la décomposition inductive des graphes Fork-Join. Tandis que l'approche *top-down* semble un peu plus complexe. Son intérêt réside dans le fait que l'on peut plus simplement l'implémenter *en place* (sans utiliser de copie). En théorie, les complexités spatiales et temporelles pour les deux algorithmes sont les mêmes. En pratique, la version *en place* est plus efficace, notamment grâce aux mécanismes de cache.

Algorithme 9 Verison *bottom-up* de la génération aléatoire uniforme d'extensions linéaires.

function RANDLINEXT-BU(P)
  if $P = \circ$ **then return** $[\,]$
  else if $P = \bullet_x$ **then return** $[x]$
  else if $P = \bullet_x \,.\, T$ **then return cons**(x, RANDLINEXT-BU(Q))
  else if $P = \square \,.\, (L \;\|\; R) \,.\, T$ **then**
    $h :=$ SHUFFLE(RANDLINEXT-BU(L), RANDLINEXT-BU(R))
    $t :=$ RANDLINEXT-BU(T)
    if $\square = \bullet_x$ **then return concat**(**cons**(x, h), t)
    else return concat(h, t)

La version *bottom-up* de l'algorithme est l'algorithme 9. On l'illustre en prenant comme exemple l'ordre de la figure 2.1 page 34. Premièrement, l'algorithme est récursivement appliqué aux deux sous-ordres en parallèle : celui contenant les d'étiquettes $\{b, d, e\}$ et celui contenant les étiquettes $\{c, f\}$. Pour ce dernier, la seule extension linéaire possible est de prendre la première étiquette c puis f résultant en l'extension linéaire (c, f). Pour la partie gauche l'étiquette b précède le *mélange uniforme* (SHUFFLE) des extensions linéaires "atomiques" (d) et (e). L'algorithme de mélange sera présenté en détail par la suite, mais ici il consiste simplement à choisir entre les extensions linéaires (d, e) ou (e, d) avec une probabilité $\frac{1}{2}$ pour les deux. En supposant le choix de (e, d), on obtient alors l'extension linéaire (b, e, d). Notons que rien n'est ajouté à la fin car le puits de ce sous-ordre est blanc. La prochaine étape consiste à mélanger (c, f) et (b, e, d) uniformément, un résultat possible étant (c, b, e, d, f) avec probabilité $\frac{1}{10}$. Alors, on concatène cette extension avec une extension du sous-ordre du bas. Par exemple, $(c, b, e, d, f, i, j, k, l)$ est une extension linéaire possible et donc uns sortie possible de l'algorithme.

Ici, le caractère *bottom-up* de l'algorithme est caractérisé par le fait que les mélanges sont faits dans les ordres les plus profonds (les feuilles de l'arbre unaire-ternaire correspondant), puis propagés étape par étape à l'ordre entier.

La version *top-down* de l'algorithme est l'algorithme 10. La principale différence avec la version *bottom-up* est que l'algorithme génère les *positions* des étiquettes dans un tableau plutôt que directement les étiquettes. L'avantage est que la plupart des opérations peuvent donc être

Algorithme 10 Verison *top-down* de la génération aléatoire uniforme d'extensions linéaires.

```
function RandLinExt-TD(P)
    function RecRandLinExt-TD(P, rankings, positions)
        if P = ∘ then return rankings
        else if P = •_x then
            rankings[x] := pop(positions)
            return rankings
        else if P = •_x . T then
            rankings[x] := pop(positions)
            return RecRandLinExt-TD(T, rankings, positions)
        else if P = □ . (L | R) . T then
            if □ = •_x then rankings[x] := pop(positions)
            upPositions := positions[ 0 ... |L| + |R| − 1 ]
            botPositions := positions[ |L| + |R| ... |P| − 1 ]
            l, r := Split(upPositions, |L|, |R|)
            rankings := RecRandLinExt-TD(L, rankings, l)
            rankings := RecRandLinExt-TD(R, rankings, r)
            return RecRandLinExt-TD(T, rankings, botPositions)
    rankings := an empty dictionary
    positions := [ 1 ... |P| ]
    return RecRandLinExt-TD(P, rankings, positions)
```

faites en place, au prix d'un niveau d'indirection supérieure. La structure $rankings$ est une table d'association entre les étiquettes et les positions dans l'extension linéaire générée. La structure $positions$ est une pile qui, initialement, contient toutes les positions disponibles entre 1 et $|P|$ (la taille de l'ordre en nombre d'étiquettes c-à-d les nœuds noirs).

Dans notre exemple, initialement, $positions$ est la liste $[1, 2, 3, 4, 5, 6, 7, 8, 9]$ et la table d'association $rankings$ est vide. Dans la première étape, la source blanche est simplement passée et les deux ensembles de positions possibles sont calculés : $upPositions$ prend les étiquettes de la partie haute de l'ordre, c-à-d $[1, 2, 3, 4, 5]$ et $botPositions$ prends le reste : $[6, 7, 8, 9]$. L'algorithme réalise alors une partition uniforme (en deux parts de bonnes tailles) des positions 1 à 5, c-à-d en les ensembles $l = \{2, 3, 4\}$ pour $\{b, d, e\}$ et $r = \{1, 5\}$ pour $\{c, f\}$. Les détails sur ce partitionnement sont donnés par la suite. Les rangs des étiquettes de chaque sous-ordre sont calculés récursivement. Encore une fois, on peut obtenir l'extension linéaire $(c, b, e, d, f, i, j, k, l)$, avec la même probabilité que dans l'algorithme précédent.

Les deux algorithmes 11 sont clairement duaux. Dans la version *bottom-up*, l'aléa vient du mélange (algorithme Shuffle) qui est un dual du partitionnement (algorithme Split) au sens d'un co-produit. Shuffle prend deux listes (ou permutations) et les mélange en une seule, tandis que Split prend une liste et la divise en deux.

L'uniformité, et donc la correction, des deux versions de l'algorithme vient de l'uniformité des algorithmes Split et Shuffle. Par exemple, il y a $\binom{5}{2} = 10$ mélanges possibles entre les ensembles $\{a, b, c\}$ et $\{d, e\}$. De manière équivalente, il y a 10 partitionnements possibles de

Algorithme 11 Algorithmes de mélange et partitionnement uniformes

```
function SPLIT(S, p, q)
    ℓ, r := [ ], [ ]
    i := 0
    v := RandomCombination(p, q)
    for all e ∈ v do
        if e then
            append S[i] to ℓ
        else
            append S[i] to r
    return ℓ, r
```

```
function SHUFFLE(ℓ, r)
    t := [ ]
    v := RandomCombination(|ℓ|, |r|)
    for all e ∈ v do
        if e then
            append pop(ℓ) to t
        else
            append pop(r) to t
    return t
```

l'ensemble $\{a, b, c, d, e\}$ en deux sous-ensembles, l'un de taille 3 et l'autre de taille 2.
Les deux algoritmes procèdent de la même façon : ils tirent aléatoirement une combinaison de p éléments parmi $p + q$, puis mélangent ou partitionnent en utilisant cette combinaison. Si l'on suppose l'uniformité de cet algorithme de génération de combinaison, on obtient alors la correction des algorithmes de génération uniforme d'extensions linéaires.

Théorème 16:
L'algorithme 9 et l'algorithme 10 génèrent aléatoirement une extension linéaire d'un ordre Série-Parallèle, de taille n, de manière uniforme. Leur complexité au pire cas est en $\Theta(n^2)$ (en nombre d'écritures mémoires). Leur complexité moyenne est asymptotiquement équivalente à :

$$\frac{1}{4}\sqrt{\frac{\pi\, n^3}{3\sqrt{2}-4}}.$$

Théorème 17 ([Möh89])**:**
Soient P un ordre Série-Parallèle et ℓ_P son nombre d'extensions linéaires. Si $P = P_1.P_2$ est la composition série de P_1 et P_2, alors $\ell_P = \ell_{P_1} \cdot \ell_{P_2}$. Si $P = P_1 \parallel P_2$ est la composition parallèle de P_1 et P_2, alors $\ell_P = \binom{n_1+n_2}{n_1} \cdot \ell_{P_1} \cdot \ell_{P_2}$, où n_1 (resp. n_2) est la taille de P_1 (resp. P_2).

Esquisse de preuve du théorème 16. La correction des deux algorithmes est facilement prouvée par une induction structurelle en utilisant le théorème 17.

Pour calculer la complexité moyenne, on procède de manière classique (cf. [FS09]).

On extrait l'asymptotique du terme général de la série génératrice F ordinaire des graphes Fork-Join obtenu par la méthode symbolique :

$$F(z) = \frac{1 - z - \sqrt{1 - 6z + z^2}}{2},$$

$$[z^n]F(z) \sim_{n\to\infty} \frac{1}{2}\sqrt{\frac{3\sqrt{2}-4}{\pi\, n^3}}\,\rho^{-n} \qquad \text{avec } \rho = 3 - 2\sqrt{2}.$$

L'asymptotique s'obtient par théorème de transfert (cf. [FO90]).

Ensuite, on définit la série génératrice bivariée C où la variable y compte le nombre d'écritures mémoires de l'algorithme et z la taille du graphe Fork-Join en entrée :

$$\begin{cases} C(z,y) = zy + zyC(z,y) + (zy+1)C(zy,y)C_r(zy,y)(C(z,y)+1) \\ C_l(z,y) = zy + zy\,C(z,y) + zyC(zy,y)C_r(zy,y)(C(z,y)+1) \\ C_r(z,y) = C_l(z,y) + C(zy,y)C_r(zy,y)C(z,y) \end{cases}$$

On calcule sa dérivée par rapport à la variable y que l'on évalue en $y = 1$:

$$\left(\frac{\partial}{\partial y}C(z,y)\right)_{|y=1} = \frac{z\left(1 + 11z - 2z^2 - (2z-3)\sqrt{11 - 6z + z^2}\right)}{4(11 - 6z + z^2)}.$$

Encore une fois, on applique le théorème de transfert de [FO90] et on en déduit le résultat. □

3.3.4 Cœur stochastique

Les deux versions de l'algorithme précédemment présenté dépendent du même *cœur stochastique* dont le point d'entrée est l'algorithme RandomCombination. Nous présentons ici l'ensemble des algorithmes permettant de répondre de manière optimale au problème de génération aléatoire d'une combinaison. Mais commençons par discuter de cette notion d'optimalité.

Mesure de complexité

Dans le contexte de la génération aléatoire et pour parler de complexité, on considère généralement un modèle de calcul tel que les machines de Turing probabiliste plutôt que le modèle plus standard de machine déterministe (et ses équivalents comme le modèle *Random Access Machine*). Une machine de Turing probabiliste est une machine de Turing déterministe possédant, en plus, un bandeau (infini) de bits aléatoires. En conséquence nous nous intéressons à cette ressource que sont les bits aléatoires.

La sortie d'un algorithme de génération aléatoire est un objet produit selon une certaine distribution. Il faut donc être capable d'estimer "la taille" d'une distribution de probabilités en nombre de bits aléatoires. Pour cela, on se base donc sur la théorie de l'information de Shannon [Sha48] et plus particulièrement sur la notion d'entropie. L'entropie est la quantité de bits minimale nécessaire à encoder une information parmi un ensemble d'informations possibles. Notre objectif, dans le reste de ce chapitre, est donc de définir un algorithme *entropique* : un algorithme qui utilise un nombre minimal de bits aléatoires.

Définition 18 (Algorithme entropique)**:**
Soit A un algorithme de génération aléatoire d'objets appartenant à un ensemble fini S, selon une mesure de probabilité μ. On dit de A qu'il est *entropique* si le nombre moyen de bits aléatoires n_e qu'il utilise pour générer un objet $e \in S$ est proportionnel à l'entropie de μ, au sens de l'entropie de [Sha48] :

$$\exists K > 0, \forall e \in S, n_e \leqslant K \cdot \sum_{x \in S} -\mu(x)\,\log_2(\mu(x)).$$

Par la suite, on appellera *constante d'entropie* une constante K vérifiant une telle inégalité.

Algorithmes

Une manière naïve de générer des combinaisons de p éléments parmi $p + q$ consiste à insérer successivement chacun des p éléments parmi les q initiaux. Dans ce cas l'algorithme utilise p entiers aléatoires de loi uniformes. Chacun de ces tirages coûtent $\mathcal{O}(\log(p + q))$ bits aléatoires [2], ce qui est bien loin de l'entropie des combinaisons de p parmi $p + q$ éléments, notamment quand p et q sont proches.

De cet algorithme naïf, on déduit qu'il faut limiter au plus l'utilisation d'entiers aléatoires de loi uniformes. L'idée clé est donc d'utiliser une loi beaucoup moins coûteuse à simuler : la loi de Bernoulli. C'est ce que propose l'algorithme 12. L'algorithme tire aléatoirement, et selon la loi uniforme, une liste l de booléens de taille $p + q$ telle que p booléens soient de valeur `True` et q de valeur `False`.

Algorithme 12 Algorithme de génération aléatoire uniforme de combinaisons

```
function RandomCombination(p, q)
    l := [ ]
    ♯True is the number of True in l
    ♯False is the number of False in l
    rndBits := a stream of random booleans produced with k-Bernoulli(p/(p+q))
    if p > log(q)^2 ∧ q > log(p)^2 then
        while ♯True <= p ∧ ♯False <= q do
            if pop(rndBits) then l := cons(True, l)
            else l := cons(False, l)
        remaining := ¬pop(l)
    else
        if p < q then
            l := a list of q times False
            remaining := True
        else
            l := a list of p times True
            remaining := False
    for i := ♯True + ♯False − 1 to p + q − 1 do
        j := uniformRandomInt[0 . . . i]
        insert remaining at position j in l
    return l
```

Malgré ce pseudo-code assez complexe, le principe de l'algorithme est assez simple. On distingue deux étapes. La première étape consiste en une boucle **while** qui remplit la liste l en tirant des booléens suivant une loi de Bernouilli de paramètre $\frac{p}{p+q}$ jusqu'à ce que l'une des deux valeurs de booléens soit suffisamment représentée (p `True` ou q `False`). La deuxième étape est

2. Pour tirer un entier aléatoire selon la loi uniforme d'un intervalle donné, il est optimal de tirer chacun des bits de la représentation binaire de cet entier, quitte à pratiquer un rejet de temps en temps. (cf. [Knu97])

une boucle **for** qui insère, comme dans l'algorithme naïf, les booléens manquant. Le reste des tests n'est là que pour des raisons techniques que nous justifierons plus tard.

Pour terminer cette description, donnons un exemple d'exécution de l'algorithme avec les entrées $p = 6$ et $q = 2$. Dans la première étape, la liste l est remplie de `True` et de `False` avec les probabilités respectives $\frac{p}{p+q}$ et $\frac{q}{p+q}$ jusqu'à ce que l'une des deux valeurs soit assez représentée. Par exemple, on pourrait obtenir la liste $l = \mathtt{F} :: \mathtt{T} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: [\,]$. Il faut alors supprimer le `False` tiré en trop (il y en a 3 au lieu de 2) : l'algorithme tire toujours une valeur en trop au cas où ce serait une des valeurs restantes à tirer (un `True` dans ce cas). Ainsi, on passe à la deuxième étape qui consiste en l'insertion uniforme de deux valeurs `True` dans la liste l. Soit, par exemple :

$$\begin{aligned} & l = \mathtt{T} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: [\,] \\ & \hookrightarrow l = \mathtt{T} :: \mathtt{T} :: \mathtt{F} :: \underline{\mathtt{T}} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: [\,] \\ & \hookrightarrow l = \mathtt{T} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: \mathtt{T} :: \mathtt{F} :: \mathtt{T} :: \underline{\mathtt{T}} :: [\,] \end{aligned}$$

Il ne reste plus qu'à retourner l qui contient bien 2 `False` et 6 `True`.

Le premier test de l'algorithme ($p > \log(q)^2 \wedge q > \log(p)^2$) consiste juste à vérifier que les entrées ne sont pas "trop petites". Dans le cas où l'une des entrée est "trop petite" par rapport à l'autre, il vaut mieux (pour économiser des bits aléatoires) directement insérer ce petit nombre de valeurs dans la liste contenant le bon nombre de l'autre valeur. Dans le détail, ce comportement est dû à un changement de régime dans la distribution des coefficients binomiaux.

Théorème 18:
L'algorithme 12 (RandomCombination) génère aléatoirement une liste de p `True` et q `False`, selon la loi uniforme. De plus il est entropique.

Démonstration. Soit l une liste générée par RandomCombination contenant p `True` et q `False`. La preuve de correction distingue deux cas :

— l est entièrement générée pendant la première étape de l'algorithme auquel cas elle est tirée avec probabilité $\left(\frac{p}{p+q}\right)^p \left(\frac{q}{p+q}\right)^q$ (le produit des $p + q$ tirage de Bernoulli de paramètres $\frac{p}{p+q}$). Cette probabilité ne dépend pas de l et est donc la même pour tout l : l est tirée uniformément parmi toutes les possibilités.

— l est tirée après $p + q - k$ tirage de Bernoulli. En utilisant le même argument que dans le premier cas, cette combinaison de $p + q - k$ valeurs est tirée uniformément. Alors, chaque valeur restante est insérée de manière uniforme, ce qui garantit l'uniformité de la liste produite : il est facile de voir qu'un mot binaire de $p + q - k$ lettres est uniformément obtenu à partir d'un mot de $p + q - k - 1$ lettres uniformément tiré dans lequel on insère une lettre.

La complexité en nombre de bits aléatoires consommés est obtenue en supposant que l'algorithme 14 (k-Bernoulli) est entropique, ce que nous démontrons par la suite. Il ne reste donc qu'à analyser le nombre de bits du flux $rndBits$ consommés et le nombre d'éléments à insérer.

Soit T la variable aléatoire comptant le nombre `True` et `False` de l à la fin de la première étape de l'algorithme. On a alors que T suit la loi suivante :

$$\mathbb{P}(T = t) = \binom{t}{p} \left(\frac{p}{p+q}\right)^{p+1} \left(\frac{q}{p+q}\right)^{t-p} + \binom{t}{q} \left(\frac{p}{p+q}\right)^{q-t} \left(\frac{q}{p+q}\right)^{q+1}$$

Pour obtenir le nombre moyens de bits consommés, on calcule l'espérance de T : $\mathbb{E}[T]$. Pour ce faire, on introduit le lemme technique suivant.

Lemme 4:
Soient p et q deux entiers tels que $q \leqslant p$ et q qui tend vers l'infini. On pose :

$$L(x) = \ln\left(\sqrt{2\pi}\sqrt{\frac{q\,(p+q)}{p}}\binom{p+q-x\sqrt{\frac{q(p+q)}{p}}}{p}\left(\frac{p}{p+q}\right)^{p+1}\left(\frac{q}{p+q}\right)^{q-x\sqrt{\frac{q(p+q)}{p}}}\right).$$

De façon formelle, il faudrait ajouter des parties entières à certains endroits de cette formule. Par soucis de lisibilité, on les omet.

Alors, pour tout x fixé, on obtient le développement suivant :

$$L(x) = -\frac{1}{2}x^2 - \frac{1}{6}\frac{x(x^2-3)}{\sqrt{q}} + o(\frac{1}{\sqrt{q}})$$

Le lemme se démontre par applications de la formule de Stirling et calcul.

Soit, en posant $t = x\sqrt{\frac{q(p+q)}{p}}$ dans l'expression de $\mathbb{P}(T=t)$ et en appliquant le lemme 4, on obtient :

$$\begin{aligned}\mathbb{P}(T=t) \sim & \frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{\frac{-1}{2}\frac{(-t+p+q)^2\,q}{(p+q)\,p}}\,\frac{\sqrt{q}}{\sqrt{(p+q)\,p}} \\ & + \frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{\frac{-1}{2}\frac{(-t+p+q)^2\,p}{q\,(p+q)}}\,\frac{\sqrt{p}}{\sqrt{q\,(p+q)}},\end{aligned}$$

quand p et q tendent vers l'infini. Ainsi, on peut calculer un équivalent asymptotique de l'espérance de T pour p et q grand :

$$\mathbb{E}[T] \sim p+q-\frac{(p+q)^{\frac{3}{2}}}{\sqrt{2pq\pi}}.$$

Ce qui signifie donc $\mathbb{E}[T] = (p+q) + o(p+q)$ quand $p > \log(q)^2$ (resp. $q > \log(p)^2$). C'est cette condition qui justifie le test au début de la première étape de l'algorithme. Il ne reste plus qu'à compter le nombre de bits aléatoires consommés et à le comparer à l'entropie de la loi uniforme sur l'ensemble des combinaisons de p parmi $p+q$ éléments, notée $E_{p,q}$ et telle que :

$$E_{p,q} = \log\binom{p+q}{p} \underset{p,q\to\infty}{\sim} (p+q)\log(p+q) - p\log(p) - q\log(q).$$

On rappelle que la génération uniforme d'un entier appartenant à l'intervalle $[0, n]$ nécessite (de manière entropique) $\Theta(\log n)$ bits aléatoires. On note $B_{p,q}$ l'entropie d'une loi de Bernoulli de paramètre $\frac{p}{p+q}$, soit, par définition :

$$B_{p,q} = -p\log\frac{p}{p+q} - q\log\frac{q}{p+q}.$$

On a donc que le nombre moyen de bits aléatoires utilisés par l'algorithme est équivalent à l'entropie de la loi uniforme sur l'ensemble des combinaisons de p éléments parmi $p + q$:

$$\mathbb{E}[T]B_{p,q} + o(p+q)\mathcal{O}(\log(p+q)) \sim E_{p,q}.$$

L'algorithme RANDOMCOMBINATION est entropique. □

Pour compléter cette preuve, il reste à démontrer qu'il existe un algorithme entropique k-BERNOULLI qui simule des variables aléatoires de Bernoulli. Avant de présenter cet algorithme, on rappelle l'algorithme, du folklore de la génération aléatoire (notamment présenté dans [Lum13]), permettant de simuler une loi de Bernoulli de n'importe quel paramètre.

Algorithme 13 Simulation d'une loi de Bernoulli

```
function BERNOULLI(p)                                   ▷ p is less than 1
    function RECBERNOULLI(a, b, p)
        if RandomBit() = 0 then
            if m > p then return False
            else RECBERNOULLI((a+b)/2, b, p)
        else
            if m < p then return True
            else RECBERNOULLI(a, (a+b)/2, p)
    return RECBERNOULLI(0, 1, p)
```

Théorème 19:
L'algorithme 13 simule une loi de Bernoulli en consommant, en moyenne, 2 bits aléatoires.

L'idée principale de l'algorithme est de pratiquer une sorte de recherche dichotomique du paramètre p dans l'intervalle $[0, 1]$. La différence est que le chemin emprunté par cette recherche dichotomique est aléatoire et donc peut ne pas trouver l'élément recherché : cela correspond à un échec de l'expérience de Bernoulli (et non un échec de la simulation).

Démonstration. La preuve de correction est directe. Il suffit de remarquer que l'algorithme parcours le préfixe (en base 2) du paramètre p et renvoie `True` avec probabilité p et `False` avec probabilité $1 - p$.

Si l'on note K la variable aléatoire comptant le nombre d'appels récursifs de l'algorithme, alors, le nombre de bits aléatoires consommés, en moyenne, est égale à l'espérance de K :

$$\mathbb{E}[K] = \sum_{k=1}^{\infty} k \cdot \frac{1}{2}^k = \frac{1}{2} \cdot \left(\frac{\mathrm{d}}{\mathrm{dz}} \frac{1}{1-z} \right) \Bigg|_{z=\frac{1}{2}} = 2$$

□

Cet algorithme consomme donc, en moyenne, un nombre constant de bits aléatoires, quel que soit le paramètre de la loi de Bernoulli à simuler. Or, l'entropie d'une loi de Bernoulli dépend

de son paramètre. Notamment, quand le paramètre est proche de 0 ou 1, l'entropie de la loi est proche de 0. Cet algorithme n'est donc, en général, pas entropique.

Au contraire, quand le paramètre est égal à $\frac{1}{2}$, l'entropie de la loi de Bernoulli est 1. L'idée, pour économiser des bits aléatoires, est donc de simuler le tirage de plusieurs variables de même loi de Bernoulli en utilisant le moins de simulations possibles de loi de paramètre $\frac{1}{2}$. C'est ce que fait l'algorithme 14.

Algorithme 14 Simulation de k variables aléatoires de loi de Bernoulli de paramètre p

```
function k-Bernoulli(p)                                   ▷ p is less than 1
    function k-BernoulliAux(p)
        k := ⌊log 1/2 / log p⌋ ,  i := 0
        v := a vector of k times True
        while ¬Bernoulli(Σ_{ℓ=0}^{i} C(k,ℓ) p^{k−ℓ} (1 − p)^ℓ) do
            i := i + 1
            j := uniformRandomInt([0 . . . k − 1])
            v[j] := False
        return v
    if p < 1/2 then return negate(k-BernoulliAux(1 − p))
    else return k-BernoulliAux(p)
```

Plus précisément, le principe de l'algorithme est de simuler des lois de Bernoulli de paramètre p par paquets. Pour ce faire, on remarque que le succès d'une simulation de loi de Bernoulli de paramètre p^k correspond aux succès de k simulations de la loi de Bernoulli de paramètre p. On choisit donc k de telle façon que p^k soit le plus proche possible de $\frac{1}{2}$, c'est-à-dire $k = \left\lfloor \frac{\log \frac{1}{2}}{\log p} \right\rfloor$. On se ramène donc au problèmes de simuler, de manière entropique, une loi de Bernoulli de paramètre proche de $\frac{1}{2}$: ce que fait l'algorithme Bernoulli.

Avant de démontrer l'efficacité d'un tel algorithme, considérons un exemple d'exécution de l'algorithme avec l'entrée $\frac{2}{7}$. Dans ce cas, $p = 1 - \frac{2}{7} = \frac{5}{7}$ et donc $k = 2$. On présente alors les différentes exécutions possibles sous forme d'un arbre de décision. On commence par simuler une loi de Bernoulli de paramètre $\left(\frac{5}{7}\right)^2$:

- si c'est un succès, cela signifie le succès de deux expériences de Bernoulli de paramètre $\frac{5}{7}$. L'algorithme retourne donc un tableau contenant deux `True` (convertis en `False`, car nous voulons simuler l'événement contraire du fait de $p = 1 - \frac{2}{7} = \frac{5}{7}$).
- si c'est un échec, cela signifie qu'au moins une des expériences de paramètres $\frac{5}{7}$ a échoué. Il faut donc simuler une nouvelle expérience pour savoir si il s'agit d'un ou de deux échecs. Dans le cas où il y a au moins un échec, l'événement où il y a exactement un échec se produit avec une probabilité de $\frac{5}{7} + (1 - \frac{5}{7}) \cdot \frac{5}{7}$, on simule donc cet événement :
  - si c'est un succès, alors il y a exactement un échec et il faut décider laquelle de la première où de la deuxième simulation est un échec. Cette décision est prise en insérant un `False` parmi les deux cases du tableau résultat.
  - sinon, cela signifie que les deux simulations sont des échecs : dans le cas où au moins deux expériences (parmi deux) sont des échecs, la probabilité que exactement

les deux soient des échecs est de $\left(\frac{5}{7}\right)^2 + 2 \cdot (1 - \frac{5}{7}) \cdot \frac{5}{7} + \left(1 - \frac{5}{7}\right)^2 = 1$: c'est nécessairement un succès. Donc l'algorithme renvoie un tableau contenant deux `False`.

Théorème 20:
L'algorithme 14, qui simule $\left\lfloor \frac{\log \frac{1}{2}}{\log p} \right\rfloor$ lois de Bernoulli de paramètre p, est entropique.

Démonstration. Soit N la variable aléatoire dénombrant le nombre d'itérations de la boucle **while** de l'algorithme k-Bernoulli. On observe que le nombre de bits aléatoires consommés est supérieurement borné par $2N + N \log k$: le nombre de bits consommés pour les simulations de loi de Bernoulli plus les bits consommés pour tiré uniformément les entiers de $[0 \ldots k]$.

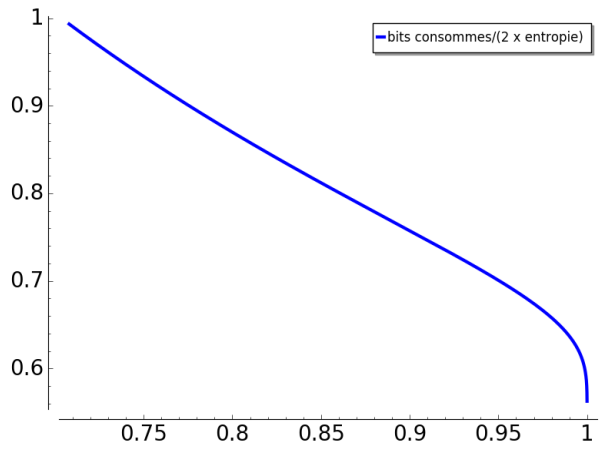
Donc, par définition, l'espérance de N est :

$$\mathbb{E}[N] = \sum_{n=0}^{k} \binom{k}{n} p^{k-n}(1-p)^n(2(n+1) + n \log k) = 2 + (1-p)(2 + \log k).$$

Il ne reste qu'à moyenner cette somme par le nombre de simulation de loi de Bernoulli de paramètre p effectuées, c'est-à-dire k. Donc le nombre moyen de bits consommés pour simuler une loi de Bernoulli de paramètre p est $\frac{2}{k} + (1-p)(2 + \log k)$. Le minimum de cette fonction de k est atteint quand k vaut $\frac{2 \log 2}{1-p}$ (on rappelle qu'ici $p > \frac{1}{2}$), autrement dit, quand le nombre moyen de bits consommés est supérieur ou égal à 2. En fait, ce cas correspond à $k = 1$, pour lequel il est préférable d'appeler l'algorithme Bernoulli. Sinon, pour $k > 1$ le nombre moyen de bits consommés vaut :

$$\mathbb{E}[N] = -2\frac{\log p}{\log 2} + (1-p) \cdot \left(2 + \log\left(-\frac{\log 2}{\log p}\right)\right).$$

On peut vérifier que ce nombre de bits est toujours inférieure à deux fois l'entropie d'une loi de Bernoulli de paramètre p tel que $\left\lfloor \frac{\log \frac{1}{2}}{\log p} \right\rfloor \geqslant 2$, comme le montre la courbe ci-dessous (rapport entre le nombre de bits consommés par l'algorithme et deux fois l'entropie) :



Cela permet de conclure que k-Bernoulli est entropique, de constante d'entropie de 2. □

3.3.5 Expérimentations

On présente ici quelques expérimentations des différents algorithmes présentés. L'implémentation des algorithmes a été faite dans le langage Ocaml.

On commence par constater que la complexité temporelle de nos algorithmes entropiques est moins bonne que l'approche "naïve" (par insertions, sans simulations de variables de Bernoulli). Théoriquement, dans les deux cas et en nombres d'écritures mémoires, l'ordre de grandeur de la complexité est le même : $\mathcal{O}(n\sqrt{n})$ où n est la taille du graphe. En pratique, les algorithmes entropiques pâtissent d'un sur-coût non compensé par la complexité du générateur de bits aléatoires qui est trop petite.

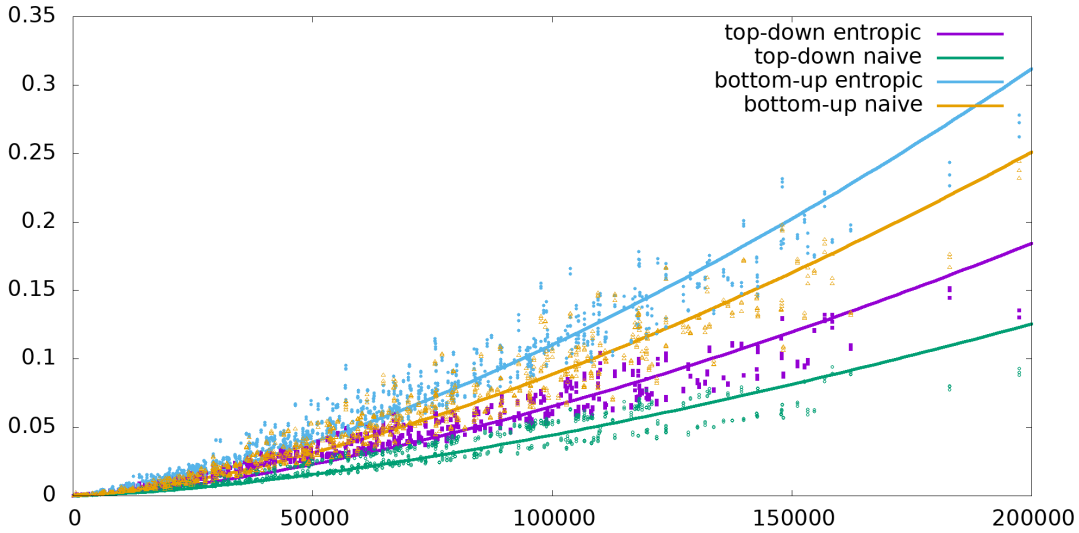


FIGURE 3.6 – Temps d'exécutions des algorithmes de génération aléatoire d'étiquetages croissants de graphes Fork-Join.

Dans la figure 3.6, on a interpolé les temps d'exécution des algorithmes *bottom-up* et *top-down* dans les cas d'une génération entropique, ou non, des combinaisons.

On observe bien que l'approche *top-down* est plus rapide car en place. On observe aussi le léger surcoût, qui n'est pas plus grand qu'un facteur 2 entre les approches naïves et entropiques. Dans de futurs travaux, nous essaierons d'implémenter plus efficacement ces différents algorithmes entropiques pour effacer ce surcoût.

Concentrons nous maintenant sur les complexités en nombre de bits aléatoires consommés.

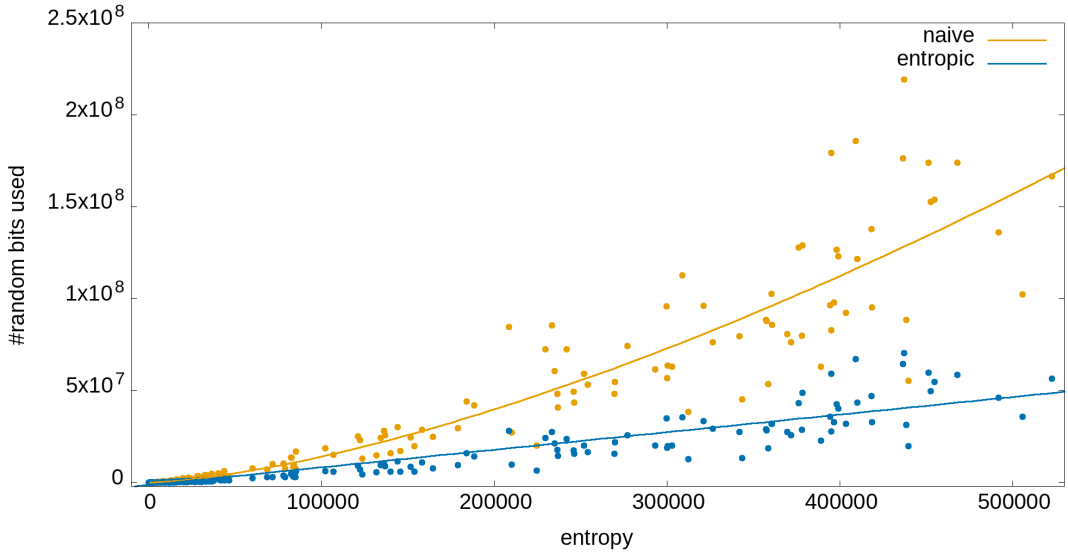


FIGURE 3.7 – Bits aléatoires consommés par les algorithmes de génération aléatoire d'étiquetages croissants de graphes Fork-Join.

La figure 3.7 confirme bien que les algorithmes entropiques consomment bien moins de bits que leur pendant naïf. Pour l'algorithme entropique le nombre de bits consommés est une fonction linéaire de l'entropie contrairement au cas naïf ($x \to x\sqrt{x}$). Dans le cas entropique, la constante d'entropie est proche de 95. Cette constante aussi importante est dû au modèle considéré : les graphes Fork-Join non-étiquetés. En moyenne ces graphes sont assez déséquilibrés : le graphe interne gauche est bien plus petit que le droit, ou l'inverse. Or ces cas correspondent aux pires pour l'algorithme RANDOMCOMBINATION, comme on le voit sur la figure 3.8.

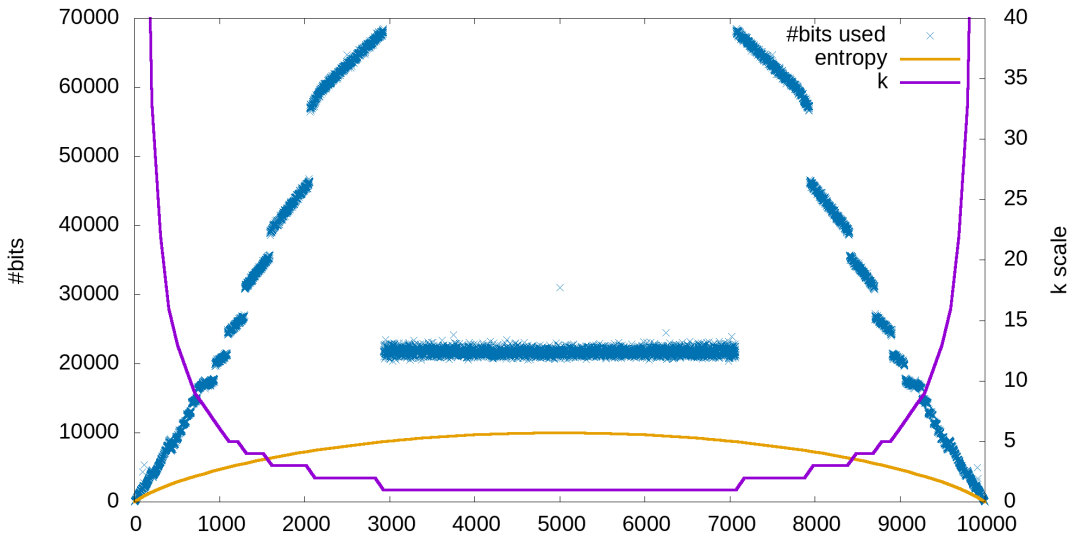


FIGURE 3.8 – Consommation de bits aléatoires de l'algorithme RANDOMCOMBINATION pour des combinaisons de 10000 éléments

Par ailleurs, on voit sur la figure 3.8 que les sauts du nuage de points correspondent à des sauts aussi observés, mais dans une moindre mesure, dans la consommation de bits de l'algorithme k-BERNOULLI (cf. figure 3.9).

Enfin, pour l'algorithme k-BERNOULLI, le nombre moyen de bits consommés est de l'ordre de 2 fois l'entropie, comme on l'avait estimé théoriquement. On le voit dans la figure 3.9.

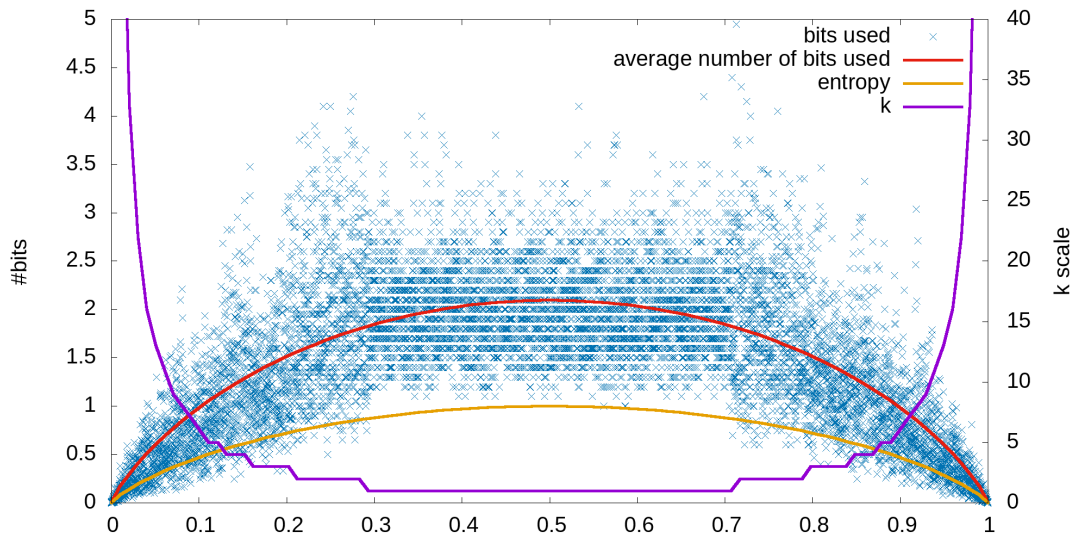


FIGURE 3.9 – Consommation de bits aléatoires de l'algorithme k-BERNOULLI

Ces expérimentations laissent entrevoir des améliorations possibles, autant du point de vue du temps d'exécution que du nombre de bits aléatoires consommés. Notamment, pour la génération de combinaisons, il serait intéressant de regarder plus précisément le phénomène qui se produit quand $p \sim \frac{q}{3}$, proportion pour laquelle le nombre de bits consommés est le plus important.

3.4 Conclusion

Dans ce chapitre, nous avons présenté deux algorithmes de génération aléatoire uniforme.

Le premier se plaçant dans le cadre de la méthode de Boltzmann et permettant de générer des structures étiquetées avec des contraintes de croissance (produit boîte). Nous avons aussi montré son efficacité pratique et une application à la génération de graphes cactus croissants.

Dans un avenir proche, nous aimerions être en mesure de générer des graphes Fork-Join croissants. Pour ce faire, on peut notamment utiliser le générateur de Boltzmann pour le produit ordonné, présenté dans [DPRS12]. Mais dans cet algorithme, comme dans ceux présentés dans ce chapitre, un problème difficile reste à résoudre pour pouvoir générer de très grandes structures (plusieurs centaines de milliers de noeuds) : l'efficacité de l'oracle. Dans leur cas comme

dans le nôtre, une grande partie du temps de calcul est prise par l'évaluation de séries génératrices en certains points. Des améliorations existantes dans le cas de générateur "classique" ([PSS12] ou [BLR15]) pourrait s'adapter à nos cas. Une autre piste serait d'étudier des méthodes d'interpolations pour obtenir des fonctions plus simples à évaluer.

Le deuxième algorithme permet de générer uniformément un étiquetage croissant d'un ordre Série-Parallèle donné. On peut donc notamment s'en servir pour étiqueter les structures générées par les générateurs de Boltzmann. En partie, il répond aussi au problème de la génération aléatoire d'exécutions de programmes Série-Parallèle.

Comme dans le cas des ordres arborescents, il reste à générer des étiquetages croissants et partiels pour pouvoir gérer les programmes non-déterministes. L'approche de [BGP13] peut, a priori, directement être adaptée à ce problème.

Enfin, il reste la question de la généralisation de ces algorithmes pour des classes d'ordres plus larges : sujet du chapitre suivant.

CHAPITRE 4

OUVERTURE : ORDRES PARTIELS SANS CYCLE

Dans ce chapitre, nous revisitons le problème du dénombrement des extensions linéaires d'un ordre donné avec un point de vue géométrique. Cette interprétation se base principalement sur les travaux de [Sta86].

En partant de cette interprétation, notre contribution est un algorithme de décomposition d'ordres partiels. Cette décomposition permet, entre autre, de caractériser des sous-classes intéressantes d'ordres partiels. Le deuxième intérêt de cette décomposition est de produire une formule symbolique dont l'évaluation donne le nombre d'extensions linéaires de l'ordre en entrée.

4.1 Interprétation géométrique des ordres partiels

Dans [Sta86], l'auteur donne une interprétation géométrique des ordre partiels, comme ensemble d'extensions linéaires. L'idée de base est de plonger un ordre partiel sur n éléments dans l'hypercube unitaire de dimension n. Pour ce faire, on part du principe que chaque élément de l'ordre correspond à une dimension de l'espace, puis l'on fait correspondre la relation d'ordre à un découpage de l'hypercube par des hyperplans.

Définition 19:
Soit $P = (E, \prec)$ un ordre partiel de taille n. Soit C l'hypercube unitaire de dimension n, défini par $C = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \forall i,\ 0 \leqslant x_i \leqslant 1\}$. Pour chaque contrainte $i \prec j \in P$ on définit le sous-ensemble convexe $S_{i,j} = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_i \leqslant x_j\}$. $S_{i,j}$ est le demi-espace obtenu en coupant $\mathbb{R}^n$ avec l'hyperplan $\{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid x_i - x_j = 0\}$.

Alors, on défini C_P, le plongement de l'ordre P dans l'hypercube C :

$$C_p = \bigcap_{i \prec j \in P} S_{i,j} \cap C$$

Remarque 7:
Notons que C_P est aussi un sous-ensemble convexe car il est défini comme une intersection d'ensembles convexes.

De la même façon que pour les ordres, on peut plonger les extensions linéaires d'un ordre dans l'hypercube unitaire en les voyant comme des ordres totaux. Ainsi, le plongement de l'ensemble des extensions linéaires d'un ordre P forme une partition du plongement de l'ordre C_p en simplexes.

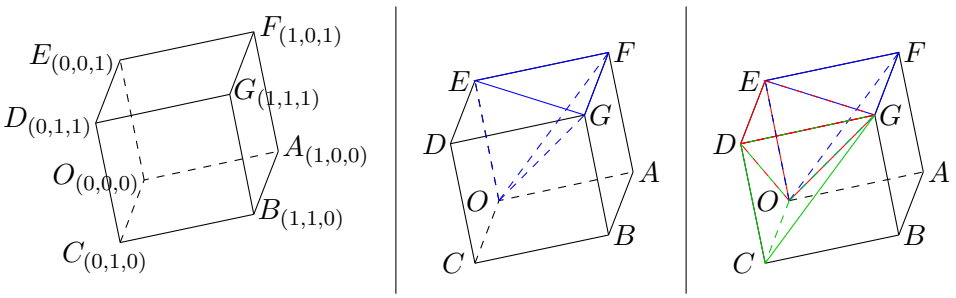


FIGURE 4.1 – De gauche à droite : l'hypercube unitaire, le plongement de l'ordre total $1 \prec 2 \prec 3$ et le plongement de l'ordre partiel $P = (\{1, 2, 3\}, \{1 \preceq 2\})$ divisé en ses trois extensions linéaires.

Un exemple de plongement d'un ordre et de ses extensions linéaires est donné dans la figure 4.1.

Étant donné que les simplexes représentant les extensions linéaires sont identiques à isométrie près, le volume du polytope C_p est donc égal aux nombres d'extensions linéaires multiplié par le volume d'un simplexe.

Fait 1 ([Sta86, Corollaire 4.2])**:**
Soit un ordre partiel P de taille n alors $|\mathscr{LE}(P)| = n! \cdot \mathrm{Vol}(C_P)$, où $\mathrm{Vol}(C_P)$ est le volume, défini par la mesure de Lebesgue, du polytope C_p : le plongement de P dans l'hypercube.

Ainsi, compter les extensions linéaires d'un ordre partiel donné revient à calculer le volume d'une union convexe de simplexes.

4.2 La décomposition *Bottom, Intermediate, Top and Cycle* (BITC)

En exploitant cette interprétation géométrique, nous présentons une décomposition agissant sur la couverture d'un ordre partiel (un DAG intransitif). Cette décomposition se veut être à la fois un outil théorique dans le cadre de l'étude du problème de comptage des extensions linéaires, mais aussi un outil algorithmique pour répondre à ce problème.

La figure 4.2 décrit les quatre règles de la décomposition. Les trois premières règles consomment toutes un nœud du DAG à décomposer : les règles B et T consomment, respectivement, un nœud avec exactement un arc entrant et un nœud avec exactement un arc sortant. La règle I consomme un nœud avec exactement un arc entrant et un arc sortant. La dernière règle

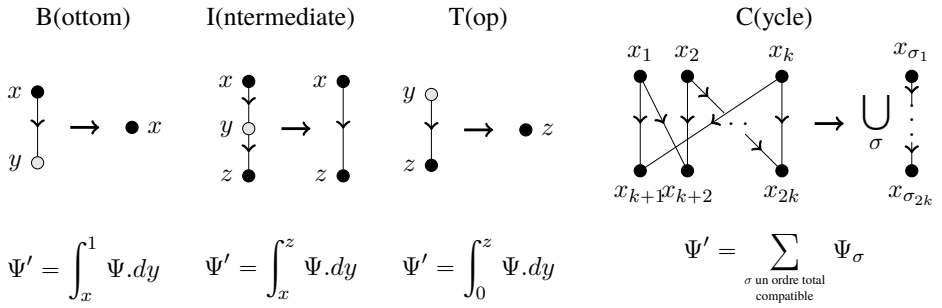


FIGURE 4.2 – La décomposition BITC

ne consomme pas de nœud mais “déplie” ce qu’on appelle des cycles de taille paire, où la notion de cycle est relative au graphe de comparabilité.

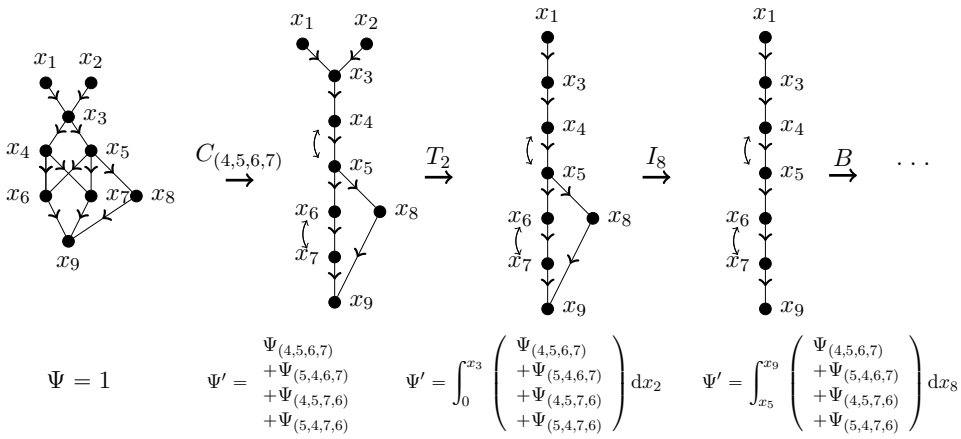
À chaque règle de décomposition correspond une formule intégrale (donnée sous la description de la règle dans figure 4.2). Ces formules permettent de calculer le nombre d’extensions linéaires du DAG à décomposer. La décomposition commence avec $\Psi = 1$, puis tant que le DAG n’est pas réduit à un nœud, une des règles de décomposition est appliquée, où Ψ' est la formule après cette étape. Quand il ne reste plus de noeud à décomposer, la formule produite est close et donc évaluable numériquement.

Théorème 21:
L’évaluation numérique de la formule symbolique construite par la décomposition BITC (multipliée par $n!$) est le nombre d’extensions linéaires du DAG décomposé.

La preuve de ce théorème repose sur deux arguments : la décomposition BITC décompose bien tous les DAGs intransitifs et les règles de calcul intégrale correspondent bien au calcul du volume du plongement.

Pour ce deuxième argument, la preuve est simple et repose sur la définition même de ce qu’est un volume. On illustre le principe de calcul sur un exemple probant.

Exemple 1 (Une décomposition BITC)**:**



Le DAG à décomposer (à gauche) est de taille 9, les nœuds étant $x_1, \ldots, x_9$. Les nœuds x_4, x_5 et x_6, x_7 forment un cycle (de taille 4) qui peut être déplié en 4 ordres totaux distincts et compatibles avec les contraintes d'ordre de ce cyle. Ainsi, la règle C décompose le DAG en 4 DAGs distincts où les éléments du cycle sont totalement ordonnés. Nous avons seulement repésenté le premier de ces 4 DAGs, mais les flèches indiquent quels éléments doivent être permutés pour obtenir les 3 autres. Notons qu'en terme d'extensions linéaires, il suffit simplement de sommer les 4 calculs restants. Maintenant, la règle T est appliquée au nœud x_2. Il est important de remarquer que cela aurait pu être fait avant l'application de la règle C. La décomposition est non-déterministe, en ce sens où il n'y a pas d'ordre privilégié d'application des règles. La règle I peut être appliquée au nœud x_8, et ainsi de suite. À la fin de cette décomposition, nous obtenons la formule symbolique suivante :

$$\begin{aligned}
Vol(P) &= Vol(P_{(4,5,6,7)}) + Vol(P_{(5,4,6,7)}) + Vol(P_{(4,5,7,6)}) + Vol(P_{(5,4,7,6)}) \\
&= \int_0^{x_3} \left(Vol(P'_{(4,5,6,7)}) + Vol(P'_{(5,4,6,7)}) + Vol(P'_{(4,5,7,6)}) + Vol(P'_{(5,4,7,6)}) \right) \mathrm{d}x_2 \\
&= \int_{x_7}^{1} \int_{x_5}^{x_9} \int_0^{x_3} \left(Vol(P'''_{(4,5,6,7)}) + Vol(P'''_{(5,4,6,7)}) + Vol(P'''_{(4,5,7,6)}) + Vol(P'''_{(5,4,7,6)}) \right) \mathrm{d}x_2 \, \mathrm{d}x_8 \, \mathrm{d}x_9 \\
&= \cdots \\
&= \int_0^1 \int_{x_1}^1 \int_{x_3}^1 \int_{x_4}^1 \int_{x_5}^1 \int_{x_6}^1 \int_{x_7}^1 \int_{x_5}^{x_9} \int_0^{x_3} 1 \, \mathrm{d}x_2 \, \mathrm{d}x_8 \, \mathrm{d}x_9 \, \mathrm{d}x_7 \, \mathrm{d}x_6 \, \mathrm{d}x_5 \, \mathrm{d}x_4 \, \mathrm{d}x_3 \, \mathrm{d}x_1 + \cdots + \cdots + \cdots \\
&= \frac{6+6+8+8}{9!} = \frac{28}{9!}.
\end{aligned}$$

Il y a donc 28 extensions linéaires distinctes dans cet exemple d'ordre partiel.

Remarque 8:
Cette approche n'est pas en contradiction avec le fait que le problème du comptage d'extensions linéaires soit $\sharp$P-complet. Par exemple, la règle cycle peut construire un terme de taille exponentielle.

4.3 Ordres partiels sans cycle

Pour en revenir à la preuve du théorème 21, nous avons besoin de montrer que la décomposition est complète, au sens où elle est capable de décomposer n'importe quels ordres partiels. Pour ce faire, nous allons montrer que les règles BIT décomposent une sous-classe d'ordre, introduites dans [DRW82] : les ordres partiels sans cycle (*cycle-free poset* en anglais). Pour conclure, il ne restera à démontrer que la règle C décompose bien les cycles.

Les ordre partiels sans cycle représentent une classe assez peu étudiée de la littérature. Pour ce qui nous intéresse, on peut citer les travaux dans [MS91] qui donnent un algorithme polynomial de reconnaissance de ces ordres. Ils sont définis par exclusion de motif.

Définition 20:
Un cycle alternant pair (de taille $2n$) est un ordre partiel sur les éléments $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$ tel que :

$$x_1 \preceq y_1, y_1 \succeq x_n, x_2 \preceq y_2, y_2 \succeq x_3, \ldots, y_{n-1} \succeq x_n, x_n \preceq y_n.$$

Soit, le motif reconnut par la règle C de la décomposition BITC.

Un ordre partiel sans cycle est un ordre ne contenant pas de cycle alternant pair.

Une autre manière de caractériser les ordres partiels sans cycle est de caractériser leur graphe de comparabilité.

Proposition 18 ([MS91])**:**
Le graphe de comparabilité d'un ordre partiel sans cycle est un graphe cordal : un graphe qui ne contient pas de cycle primitif [1] de taille plus grande que 3.

Remarque 9:
Les cycles primitifs rencontrés dans le graphe de comparabilité d'un ordre partiel sont nécessairement de taille paire. Pour s'en convaincre, il suffit de remarquer qu'il n'existe pas d'orientation transitive d'un cycle de taille impaire et donc d'ordre partiel associé.

Théorème 22:
La décomposition BIT (sans la règle C) caractérise en temps linéaire, en la taille de l'ordre partiel considéré, la classe des ordres partiels sans cycle.

On dit que la couverture un ordre partiel sans cycle est BIT-décomposable.

Un intérêt de la BIT décomposition est qu'elle est beaucoup plus élémentaire que d'autres algorithmes de reconnaissance (par exemple [MS91]). Elle a la même complexité linéaire, modulo réduction transitive.

Avant de prouver ce théorème, on rappelle la définition de clique maximale, notion essentielle de la preuve.

Définition 21 (Clique maximale)**:**
Soit un graphe non dirigé $G = (V, E)$. Une clique $C \subseteq V$ est un ensemble de nœuds tel que toutes les paires $(u, v) \in C \times C$ sont des arêtes de G. C'est une clique maximale si elle n'est contenue dans aucune autre clique.

Remarque 10:
Une clique maximale du graphe de comparabilité d'un ordre partiel correspond à un chemin entre une source et un puits de son DAG couvrant.

Démonstration. On procède par induction sur le nombre d'éléments de l'ordre partiel car chaque règle BIT supprime exactement un tel élément. Notre objectif est de montrer que si aucune des règles BIT ne peut être appliquée, alors l'ordre partiel considéré contient un cycle. Un cas spécial est que nous considérons la décomposition terminée quand l'ordre ne contient plus qu'un seul élément. Pour simplifier nous ne considérons que les ordres partiels connexes (dont la couverture est connexe), la généralisation au cas non connexe étant trivial. Comme cas de base, il est aisé de constater que tous les ordres de taille au plus 3 sont sans cycle et BIT-décomposables.

L'idée clé de cette preuve est d'interpréter les règles BIT en terme d'intersection des cliques maximales du graphe de comparabilité. Voici quelques observations préliminaires :

— si $x \preceq y \preceq z$ sont en série, alors chaque clique maximale contenant x contient nécessairement y et z, et de même pour y et z ;
— si x est un nœud "pendant", attaché à l'ordre par un autre nœud y, alors chaque clique maximale contenant x contient aussi y.

1. Un cycle primitif n'est contenu par aucun autre cycle.

Notre hypothèse d'induction est que la décomposition BIT caractérise les ordres sans cycle de taille $n-1$. Montrons maintenant que c'est aussi le cas pour les ordres de taille n. Soient C_v l'ensemble des cliques maximales du graphe de comparabilité contenant v et $I_v = \bigcap_{C \in C_v} C$ (l'intersection des cliques maximales contenant v). Observons que $\forall v,\ \{v\} \subseteq I_v$. Considérons donc les cas suivants :

1. si $\forall v,\ I_v = \{v\}$ alors aucune des règles BIT ne s'applique car les précédentes observations impliquent que l'application des règles nécessite au moins deux éléments dans l'intersection. Montrons que dans ce cas l'ordre contient un cycle de taille au moins 4. Premièrement, on peut trouver deux sources distinctes, ou si il n'y a qu'une source, deux voisins incomparables de cette source. Autrement, au moins une des règles s'appliquerait. Le même principe s'applique aux puits. Et puisque que les puits et sources sont connectés deux à deux, nous obtenons un cycle de taille au moins 4. Ainsi, l'ordre contient un cycle.
2. autrement, si $\exists u,\ u \neq v,\ I_v = \{v, u\}$, considérons deux sous cas :
   — si v (ou u) est un nœud pendant alors la règle B ou T s'applique et fait décroître la taille de l'ordre sans créer de cycle. Nous considérons alors un ordre de taille $n-1$ dont la caractérisation est décidée, par hypothèse d'induction.
   — sinon, aucune des règles BIT ne s'appliquent. Cela signifie que (v, u) (ou u, v)) est un isthme de la couverture (un arc dont l'élimination entraîne déconnexion du graphe). On remarque qu'il est impossible que tous les autres noeuds soient dans ce cas là (sans violer l'hypothèse de connexité). On considère donc un autre noeud.
3. dans les cas où $\exists v,\ |I_v| \geqslant 3$, deux cas sont possibles :
   — un des nœuds de I_v est en série et donc la règle I s'applique. Nous pouvons donc utiliser l'hypothèse d'induction pour conclure.
   — si la règle I ne s'applique à aucun des noeuds de I_v, cela signifie que chacun des noeuds appartient à plusieurs chaînes de l'ordre et donc qu'il faut choisir un autre candidat. On remarque encore que tous les noeuds ne peuvent pas être dans ce cas là (sans violer l'hypothèse de connexité). On considère donc un autre noeud.

□

Corollaire 2:
Tout DAG intransitif est BITC décomposable.

Démonstration. Aucune des règles BIT ne crée pas de cycle. Tous les cycles peuvent être consommés par la règle C. Donc, après toutes les applications possibles de la règle C, il ne reste plus qu'à décomposer un ensemble d'ordres sans cycle avec les règles BIT. □

Concluons cette étude de la décomposition BITC, par la proposition suivante qui donne une piste pour une étude plus approfondie de la complexité, au sens algorithmique, de la décomposition BITC.

Proposition 19:
Soit un ordre partiel P de taille n. L'expression symbolique Ψ construite à l'aide la décomposition BITC a une taille exponentielle en n. Précisément, chaque cycle de taille $2k$ donne lieu à la construction de $(k!)^2 \cdot 2^{k(k-2)}$ ordres de taille n.

Si P est un ordre sans cycle, la taille de Ψ est linéaire en n, en nombre de variables et en nombre de symbole d'intégrale.

Démonstration. Pour le cas des ordres sans cycle, la preuve est une simple induction : Les règles BIT génèrent exactement un symbole intégral et une nouvelle variable pour chaque nœud consommé.
Supposons que le DAG contienne un cycle de taille $2k$ partitionné en deux partitions p_1 et p_2, où les arcs sont orientés de p_1 vers p_2. Pour compter le nombre d'extensions linéaires compatibles avec ce cycle nous commençons par compter le nombre d'extensions linéaires du graphe dirigé, biparti et complet $K_{k,k}$ des deux partitions p_1 et p_2. Ce graphe a $(k!)^2$ extensions linéaires, et si nous supprimons un arc entre entre deux nœuds $u \in p_1$ et $v \in p_2$, ce nombre est doublé : on peut échanger la position de u et v dans toutes les extensions linéaires. Or, il faut supprimer $k(k-2)$ arcs pour retrouver le cycle initial ($k-2$ arcs sont supprimés par nœuds pour chacun des k nœuds de p_1, puisque 2 arcs par nœuds sont nécessaires pour former un cycle). Ainsi, nous construisons $2^{k(k-2)}$ extensions linéaires à partir des $(k!)^2$ extensions linéaires de $K_{k,k}$. Donc, la formule symbolique résultante de l'application de la règle C a un total de $(k!)^2 2^{k(k-2)}$ composantes. □

4.4 Expérimentation

Dans cette section nous présentons une expérimentation basée sur la décomposition BITC. L'objectif est de vérifier si cette décomposition a un quelconque intérêt pratique. Pour cela nous avons récupérer des bases de données de DAGs issues des domaines de l'ordonnancement (TaskDAGs, cf. [JCB00]) et du dessin de graphes (DAG-random [2]). Les plus petits Dags ont seulement 10 noeuds et les plus grands dépassent les 400.

Base de données	**♯DAGs**	**♯sans cycle**	**Comptage** (max 30s)
TaskDAGs	1297	740 ($\approx 57\%$)	737 ($\approx 57\%$)
DAG-random	910	727 ($\approx 80\%$)	413 ($\approx 45\%$)

Table 4.1 – DAG databases benchmark

Les résultats sont présentés dans la table 4.1. Pour les deux bases une proportion importante des DAGs sont BIT-décomposables. Pour ces DAGs en particulier nous avons autorisé au maximum 30 secondes pour calculer le nombre d'extensions linéaires selon la formule symbolique obtenue par la décomposition. Les calculs numériques sont délégués au logiciel de calcul formel Maxima. Au-delà d'une centaine de noeuds, les 30 secondes ne suffisent plus et accorder plus de temps ne change pas les résultats. Il est donc difficile de juger de la nature de la complexité du calcul numérique sous-jacent.

2. `http://www.graphdrawing.org/data.html`

Conclusion et Perspectives

Dans ce manuscrit nous avons exploré la combinatoire d'une famille de programmes concurrents dont la structure de contrôle correspond à un DAG. Nous avons utilisé des outils de combinatoire analytique pour obtenir des résultats quantitatifs sur ces programmes, notamment sur leur expressivité, en calculant l'asymptotique du nombre moyen d'exécutions possibles d'un programme "diamant" ou Série-Parallèle. Dans ce cadre, nous avons notamment développé des outils dont la portée dépasse le cadre de la concurrence : produits ordonné et coloré.

D'un point de vue pratique nous avons élaboré des algorithmes de génération aléatoire uniforme à la fois pour générer des exécutions de programmes et également pour générer des programmes concurrents. Si ces algorithmes ne sont pas encore applicables à des programmes réels, ils représentent des briques élémentaires qui nous permettent de l'envisager à plus long terme.

Perspectives

Étant donné les résultats présentés dans ce manuscrit, on peut dégager plusieurs pistes de recherche concernant la combinatoire des programmes concurrents et des structures croissantes. Dans cette section, nous détaillons les différents problèmes que nous aimerions résoudre dans un avenir plus ou moins proche.

Ordres partiels sans cycle

Complexité Dans [BH89] les auteurs conjecturent que le problème de comptage des extensions linéaires d'un ordre sans cycle est de complexité polynomiale. Nous avons montré que ce calcul pouvait se faire en évaluant une formule intégrale de taille linéaire en la taille de l'ordre en entrée. Même si ce résultat est encourageant, d'autres résultats semblent suggérer que le problème est dur (est-il $\sharp$P-complet ?). Dans [BBDL$^+$11] un théorème statue que le problème d'intégrer un polynôme quelconque en n variables, sur un simplexe est NP-difficile.

Néanmoins, dans notre cas les polynômes sont homogènes et d'autres résultats de cet article peuvent (peut-être) être utilisés pour ce cas particulier.

Mais, le degré du polynôme et son nombre de variables semblent être les paramètres critiques du problème. Or, dans notre cas, le nombre de variables du polynômes et son degré correspondent à la taille de l'ordre.

Avec ces résultats en tête, il nous semble réalisable à moyen terme de trancher la question de la complexité du comptage des extensions linéaires des ordres sans cycles.

De manière complémentaire, une partie de la complexité du problème de dénombrement des extensions linéaires semble provenir de la structure de cycle, et plus généralement, de celle de couronne. Dans l'article [BW91], qui montre la difficulté de ce problème, les auteurs conjecturent la difficulté du problème pour les ordres de hauteur 2, qui contiennent les cycles et couronnes. Il nous semble intéressant d'aborder à plus long terme cette question.

Combinatoire D'un point de vue plus combinatoire, et particulièrement celui de la combinatoire analytique, la structure des ordres sans cycle semble être très compliquée à analyser. Pour le moment les deux idées pour mener cette étude sont :

— de partir du modèle d'ordre Série-Parallèle (graphes Fork-Join) et de le complexifier en ajoutant des paramètres,

— de partir d'un modèle d'arbres croissants étiquetés avec autorisation des répétitions d'étiquettes pour signifier les synchronisations entre processus.

Un autre objet d'étude intéressant est celui des graphes de comparabilité. Pour le moment, notre point de vue a été de considérer les couvertures des ordres et non leur graphe de comparabilité. Or, vus les travaux assez récents sur plusieurs sous-classes des graphes cordaux (réseaux apolloniens [SD07], k-arbres [DHS16]), cette idée peut être le point de départ de nouvelles études.

Produit ordonné et coloré

D'un point de vue combinatoire, ces opérateurs permettent de spécifier des structures qui n'étaient pas spécifiables jusqu'à présent dans la méthode symbolique. Nous devons explorer l'expressivité de ces opérateurs.

Du point de vue de la combinatoire analytique, il reste encore beaucoup à comprendre de ces opérateurs pour pouvoir les utiliser simplement. Par exemple, dans le cas des graphes Fork-Join monochromes et croissants, nous aimerions comprendre les phénomènes en jeu au niveau des fonctions génératrices. Où sont exactement les singularités et quelles sont leur type ? Peut-on obtenir les termes des ordres inférieurs dans le développement asymptotique ?

Enfin, il nous reste aussi à étudier des paramètres inductifs des graphes Fork-Join croissants, tels que la profondeur de branchement maximale moyenne ou le nombre de noeuds blancs moyens.

Concurrence

Non-déterminisme Une des perspectives à court terme est d'adapter les résultats de [BGP13] sur le non-déterminisme dans les processus arborescents aux processus Série-Parallèle décrit dans ce manuscrit. Nous pensons que la même technique d'étiquetage partiel peut encore mener à des algorithmes efficaces pour la génération aléatoire d'exécutions non-déterministes.

Applications Une des motivations principales de ce travail concerne l'apport des algorithmes de comptage et de génération aléatoire aux différentes techniques de *model checking*. Il serait donc intéressant de les expérimenter dans ce cas et donc d'implémenter un *model checker proof-of-concept*.

Itération Le formalisme concurrent que nous étudions dans cette thèse a été conçu dans l'idée de facilement être analysable (combinatoirement). De ce fait, il ne contient pas de construction itérative (boucles) ou récursive (appels récursifs). L'ajout de telles constructions rapprocherait beaucoup notre modèle de programmes plus réalistes.

Bisimulation Nous avons considéré que deux programmes étaient équivalents si leurs ordres partiels induits étaient isomorphes. Or, en théorie de la concurrence, il existe des notions plus fines d'équivalence entre processus, par exemple la bisimilarité ou l'équivalence de traces. D'un point de vue pratique, on peut voir ces relations d'équivalence comme une sorte de compression des comportements.

Une étude quantitative de ces différentes relations d'équivalence notamment en terme de taux de compression, nous semble très intéressante.

Bibliographie

[ACRS16] Umut Acar, Arthur Charguéraud, Mike Rainey, and Filip Sieczkowski. Dag-calculus : A calculus for parallel computation. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 18–32, 2016.

[AM15] Samy Abbes and Jean Mairesse. Uniform generation in trace monoids. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, pages 63–75, 2015.

[AS64] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions : with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation, 1964.

[BBDL+11] Velleda Baldoni, Nicole Berline, Jesus De Loera, Matthias Köppe, and Michèle Vergne. How to integrate a polynomial over a simplex. *Mathematics of Computation*, 80(273) :297–325, 2011.

[BD99] Russ Bubley and Martin Dyer. Faster random generation of linear extensions. *Discrete mathematics*, 201(1) :81–88, 1999.

[BDF+16] Olivier Bodini, Matthieu Dien, Xavier Fontaine, Antoine Genitrini, and Hsien-Kuei Hwang. Increasing diamonds. In *Latin American Symposium on Theoretical Informatics*, pages 207–219. Springer, Berlin, Heidelberg, 2016.

[BDGP17] Olivier Bodini, Matthieu Dien, Antoine Genitrini, and Frédéric Peschanski. The ordered and colored products in analytic combinatorics : Application to the quantitative study of synchronizations in concurrent processes. In *2017 Proceedings of the Fourteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 16–30. Society for Industrial and Applied Mathematics, 2017.

[Ben75] E. A. Bender. An asymptotic expansion for the coefficients of some formal power series. *Journal of the London Mathematical Society*, s2-9(3) :451–458, 1975.

[BFS92] F. Bergeron, P. Flajolet, and B. Salvy. Varieties of Increasing Trees. In *CAAP*, volume 581 of *LNCS*, pages 24–48. Springer, 1992.

[BGP13] Olivier Bodini, Antoine Genitrini, and Frédéric Peschanski. The combinatorics of non-determinism. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 24. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[BGP16] Olivier Bodini, Antoine Genitrini, and Fréderic Peschanski. A Quantitative Study of Pure Parallel Processes. *Electronic Journal of Combinatorics*, 23(1) :P1.11, 2016.

[BGPR15] O. Bodini, A. Genitrini, F. Peschanski, and N. Rolin. Associativity for Binary Parallel Processes : A Quantitative Study. In *Algorithms and Discrete Applied Mathematics - First International Conference, CALDAM'15*, pages 217–228, 2015.

[BH89] V. Bouchitte and M. Habib. *The Calculation of Invariants for Ordered Sets*, pages 231–279. Springer Netherlands, Dordrecht, 1989.

[BLL98] François Bergeron, Gilbert Labelle, and Pierre Leroux. *Combinatorial species and tree-like structures*. Cambridge University Press, 1998.

[BLR15] Olivier Bodini, Jérémie Lumbroso, and Nicolas Rolin. Analytic samplers and the combinatorial rejection method. In *Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2015, San Diego, CA, USA, January 4, 2015*, pages 40–50, 2015.

[BMS17] Nicolas Basset, Jean Mairesse, and Michèle Soria. Uniform sampling for networks of automata. In *CONCUR 2017 28th International Conference on Concurrency Theory*, Berlin, Germany, September 2017.

[Bod10] O. Bodini. *Autour de la génération aléatoire sous modèle de Boltzmann*. Habilitation thesis, UPMC, 2010.

[BRS12] Olivier Bodini, Olivier Roussel, and Michele Soria. Boltzmann samplers for first-order differential specifications. *Discrete Applied Mathematics*, 160(18) :2563–2572, December 2012. 15 pages.

[BW91] Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3) :225–242, 1991.

[Dav62] Harold Thayer Davis. *Introduction to Nonlinear Differential and Integral Equations*. Courier Corporation, 1962.

[dBKR71] Nicolaas Govert de Bruijn, Donald Ervin Knuth, and SO Rice. The average height of planted plane trees. Technical report, DTIC Document, 1971.

[DFLS04] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4-5) :577–625, 2004.

[DHS16] Alexis Darrasse, Hsien-Kuei Hwang, and Michele Soria. Shape measures of random increasing k-trees. *Combinatorics, Probability and Computing*, 25(5) :668–699, 2016.

[DLDMDB06] Karel De Loof, Hans De Meyer, and Bernard De Baets. Exploiting the lattice of ideals representation of a poset. *Fundamenta Informaticae*, 71(2, 3) :309–321, 2006.

[DPRS12] Alexis Darrasse, Konstantinos Panagiotou, Olivier Roussel, and Michele Soria. Biased boltzmann samplers and generation of extended linear languages with shuffle. In *23rd International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA'12)*, pages 125–140. Discrete Mathematics and Theoretical Computer Science, 2012.

[DRW82] D Duffus, I Rival, and P Winkler. Minimizing setups for cycle-free ordered sets. *Proceedings of the American Mathematical Society*, 85(4) :509–513, 1982.

[Duf65] Richard J Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2) :303–318, 1965.

[EW08] Byron Ellis and Wing Hung Wong. Learning causal bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103(482) :778–789, 2008.

[FO90] Philippe Flajolet and Andrew Odlyzko. Singularity analysis of generating functions. *SIAM Journal on discrete mathematics*, 3(2) :216–240, 1990.

[FS93] Philippe Flajolet and Michèle Soria. General combinatorial schemas : Gaussian limit distributions and exponential tails. *Discrete Mathematics*, 114(1-3) :159–180, 1993.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. cambridge University press, 2009.

[GDG+08] M.-C. Gaudel, A. Denise, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet. Coverage-biased random exploration of models. *Electronic Notes in Theoretical Computer Science*, 220(1) :3 – 14, 2008. Proceedings of the Fourth Workshop on Model Based Testing (MBT 2008).

[Gen17] Antoine Genitrini. Combinatoire énumérative et analytique en logique propositionnelle et en théorie de la concurrence :vers une quantification de l'expressivité des modèles. Habilitation thesis, Université Pierre et Marie Curie, 2017.

[Gre83] Daniel Hill Greene. Labelled formal languages and their uses. Phd thesis, 1983.

[GS05] Radu Grosu and Scott A Smolka. Monte carlo model checking. In *TACAS*, volume 3440, pages 271–286. Springer, 2005.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8) :666–677, 1978.

[Hub06] Mark Huber. Fast perfect sampling from linear extensions. *Discrete Mathematics*, 306(4) :420–428, 2006.

[JCB00] A. Jarry, H. Casanova, and F. Berman. Dagsim : a simulator for dag scheduling algorithms. Technical report, ENS Lyon, 2000.

[JVV86] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43 :169–188, 1986.

[KK91] Alexander Karzanov and Leonid Khachiyan. On the conductance of order markov chains. *Order*, 8(1) :7–15, 1991.

[KM12] Arnold Knopfmacher and Augustine O Munagi. Permutations with interval cycles. *Ars Comb.*, 105 :273–288, 2012.

[KMM02] Daniel Krob, Jean Mairesse, and Ioannis Michos. *On the Average Parallelism in Trace Monoids*, pages 477–488. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.) : Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Knu98] D. E. Knuth. *The art of computer programming, volume 3 : (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[KP16] Markus Kuba and Alois Panholzer. Combinatorial families of multilabelled increasing trees and hook-length formulas. *Discrete Mathematics*, 339(1) :227–254, 2016.

[Law13] Derek F. Lawden. *Elliptic Functions and Applications*. Springer Science & Business Media, 2013.

[Lum13] Jérémie Lumbroso. Optimal discrete uniform generation from coin flips, and applications. *CoRR*, abs/1304.1916, 2013.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Möh89] Rolf H Möhring. Computationally tractable classes of ordered sets. In *Algorithms and order*, pages 105–193. Springer, 1989.

[MS91] Tze-Heng Ma and Jeremy P Spinrad. Cycle-free partial orders and chordal comparability graphs. *Order*, 8(1) :49–61, 1991.

[MS15] István Miklós and Heather Smith. Sampling and counting genome rearrangement scenarios. *BMC Bioinformatics*, 16(14) :S6, Oct 2015.

[MT00] George Marsaglia and Wai Wan Tsang. The ziggurat method for generating random variables. *Journal of statistical software*, 5(8) :1–7, 2000.

[ODG+11] Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne, and Sylvain Peyronnet. Uniform monte-carlo model checking. In *Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 127–140, 2011.

[PEB07] D. Potop-Butucaru, S. A. Edwards, and G. Berry. *Compiling Esterel*. Springer, 2007.

[Pet62] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390, 1962.

[PR94] Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2) :373–386, 1994.

[PSS12] Carine Pivoteau, Bruno Salvy, and Michele Soria. Algorithms for combinatorial structures : Well-founded systems and newton iterations. *Journal of Combinatorial Theory, Series A*, 119(8) :1711–1773, 2012.

[SD07] Michèle Soria and Alexis Darrasse. Degree distribution of random apollonian network structures and boltzmann sampling. *Discrete Mathematics & Theoretical Computer Science*, 2007.

[Sed08] Robert Sedgewick. Left-leaning red-black trees. In *Dagstuhl Workshop on Data Structures*, page 17, 2008.

[Sen07] Koushik Sen. Effective random testing of concurrent programs. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 323–332. ACM, 2007.

[Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3) :379–423, July 1948.

[SI09] M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *2009 IEEE 25th International Conference on Data Engineering*, pages 317–328, March 2009.

[Sta74] Richard P Stanley. Enumeration of posets generated by disjoint unions and ordinal sums. *Proceedings of the American Mathematical Society*, 45(2) :295–299, 1974.

[Sta86] Richard P Stanley. Two poset polytopes. *Discrete & Computational Geometry*, 1(1) :9–23, 1986.

[Tak91] Lajos Takács. Conditional limit theorems for branching processes. *International Journal of Stochastic Analysis*, 4(4) :263–292, 1991.

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8) :103–111, August 1990.

[VTL79] Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 1–12, New York, NY, USA, 1979. ACM.

[YCER11] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in c compilers. In *ACM SIGPLAN Notices*, volume 46, pages 283–294. ACM, 2011.

ANNEXE A

ORDRES PARTIELS

Définition 22 (Ordre partiel)**:**
Un ordre partiel P est un couple composé d'un ensemble G d'éléments distincts quelconques et d'une relation binaire $\prec$:

— antisymétrique : $\forall a, b \in G, a \preceq b \Rightarrow \neg(b \preceq a)$,
— réflexive : $\forall a \in G, a \preceq a$,
— transitive : $\forall x, y, z \in G x \preceq y \wedge y \preceq z \Rightarrow x \preceq z$

Quelques exemples d'ordre partiels "de la vie courante" sont :

— $\mathbb{Z}$ muni de la relation d'ordre "plus petit ou égal" est un ordre partiel,
— la relation d'inclusion est un ordre partiel sur l'ensemble de tous les ensembles,
— la relation "est préfixe de" est un ordre partiel sur l'ensemble de tous les mots d'un même alphabet.

D'un point de vue mathématique un ordre partiel est un ensemble d'éléments muni d'une relation binaire. Or, la transitivité de cette relation ajoute beaucoup d'information (au pire $\mathcal{O}\left(n^2\right)$ arêtes dans la représentation graphique). Pour éviter ce problème, une manière classique de travailler avec des ordres partiels est d'utiliser leur graphe couvrant ou *couverture*.

Définition 23 (Graphe couvrant ou couverture)**:**
Soit $(P, \preceq)$ un ordre partiel. La couverture G de P est un graphe dirigé acyclique (DAG) dont l'ensemble des nœuds est l'ensemble de P et l'ensemble des arc est tel que

$$\{(x, y) \in P^2 \mid (x \preceq y) \wedge \neg(\exists z \in P,\ x \preceq z \preceq y)\}.$$

Fait 2:
L'ensemble des couvertures est en bijection avec l'ensemble des ordres partiels. De plus, l'ensemble des couvertures est l'ensemble des graphes dirigés acycliques (DAG en anglais) intransitifs.

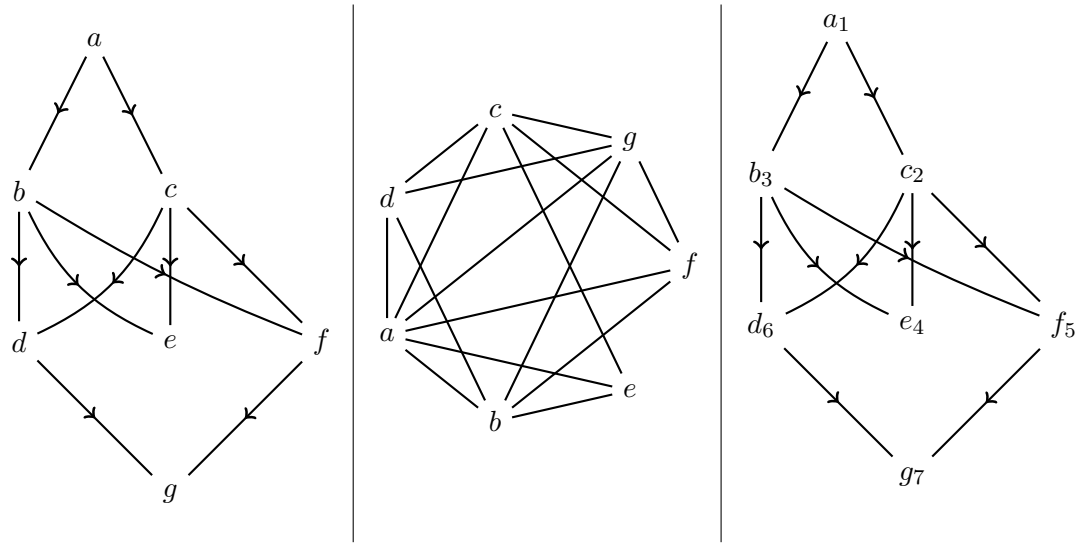


FIGURE A.1 – Exemple de couverture d'un ordre partiel ; de son graphe de comparabilité ; et d'un des étiquetages croissants de sa couverture.

Par exemple, le graphe dirigé sur la gauche de la figure A.1 est la couverture de l'ordre partiel défini sur l'ensemble $\{a, b, c, d, e, f, g\}$ muni de la relation d'ordre $\{a \preceq b, a \preceq c, b \preceq d, c \preceq d, b \preceq e, c \preceq e, d \preceq g, c \preceq f, f \preceq g\}$ fermée transitivement. Pour observer la fermeture transtitive de l'ordre, on utilisera plutôt son graphe de comparabilité :

Définition 24 (Graphe de comparabilité)**:**
Le graphe de comparabilité d'un ordre partiel P est le graphe, non dirigé, transitif $G = (V, E)$ où V est l'ensemble des éléments de l'ordre partiel et E est l'ensemble des arêtes, défini par $E = \big\{(x, y) \in V^2 \mid x \preceq y \vee y \preceq x\big\}$.

Le graphe au milieu de la figure A.1 est le graphe de comparabilité de l'ordre partiel pris en exemple précédemment.

Par la suite, on se permettra d'omettre les directions des arcs dans les représentations graphiques des couvertures, en prenant la convention que les arcs sont dirigés de bas en haut.

Définition 25 (Extension linéaire)**:**
Soit $(P, \preceq)$ un ordre partiel. Une extension linéaire est un ordre total et strict $(P, \prec)$ (une relation binaire, antisymétrique, transitive et non réflexive) sur les éléments de l'ensemble de P et compatible avec la relation de P :

$$\forall x, y \in P,\ (x \prec y) \wedge (y \prec y) \wedge (x = y)$$
$$\forall x, y \in P,\ x \neq y \wedge x \preceq y \implies x \prec y$$

On note $\mathscr{LE}(P)$ l'ensemble des extensions linéaires de P.

En reprenant l'ordre donné en exemple dans la figure A.1, une extension linéaire de ses extensions linéaires est : $a \prec c \prec b \prec e \prec f \prec d \prec g$.

Fait 3:
Soit un ordre partiel P. Chaque extension linéaire de P correspond à un tri topologique des noeuds de la couverture de P.

Définition 26 (Plongement combinatoire)**:**
Soit un graphe dirigé G. Un plongement combinatoire de G est le graphe G dont chaque noeud est décoré par une permutation de ses arcs sortants.

Plus intuitivement, le plongement combinatoire d'un graphe donne un ordre à ses arcs sortants. On peut alors désigner les arcs sortants d'un noeud de l'arc "gauche" à l'arc "droit".

Quand on travaille sur les graphes, on fait naturellement un choix sur l'ordre de leurs arcs, que ce soit quand on les représente graphiquement ou quand ils sont parcourus par un algorithme. Ce choix correspond à un plongement combinatoire du-dit graphe.

Pour conclure cette section et bien comprendre l'approche utilisée dans nos travaux, on fait le lien entre extensions linéaires et étiquetages croissants de la couverture d'un ordre partiel.

Définition 27 (Étiquetage croissant)**:**
Un étiquetage d'un graphe dirigé acyclique $G = (V, A)$, où V est l'ensemble de ses noeuds et A l'ensemble de ses arcs, est une bijection de V dans l'intervalle d'entier $[1, \ldots, |V|]$.

Un étiquetage croissant de G, est un étiquetage de G tel que **toute** les suites d'étiquettes le long d'un chemin d'une source à un puits de G soient croissantes.

Théorème 23:
Étant donné un ordre partiel P, l'ensemble de ses extensions linéaires de P est en bijection avec l'ensemble des étiquetages croissants de sa couverture.

Démonstration. Étant donné une extension linéaire $(x_0, \ldots, x_{n-1})$ de P, on construit de manière unique un étiquetage croissant de sa couverture en associant l'étiquette i au noeud x_i. Cette construction est trivialement surjective. De même, elle est clairement injective : si deux étiquetages croissants sont différents alors leurs extensions linéaires correspondantes le sont aussi. C'est donc une bijection. □

La partie gauche de la figure A.1 donne un étiquetage croissant correspondant à l'extension linéaire $a \preceq c \preceq b \preceq e \preceq f \preceq d \preceq g$.

ANNEXE B

Transformées de Laplace et de Borel

Ici, nous rappelons les relations classiques entre les transformées de Laplace et de Borel usuelles et leur versions combinatoire.

Par définition, la transformée de Laplace usuelle est définie par $\mathcal{L}f = \int_0^\infty \exp(-zt)f(t)dt$ à la place de $\mathcal{L}_c f = \int_0^\infty \exp(-t)f(zt)dt$. Cet opérateur est clairement linéaire. Par un simple changement de variable on obtient $\mathcal{L}f(z) = \dfrac{1}{z}\left(\mathcal{L}_c f\right)\left(\frac{1}{z}\right)$ ou de manière équivalente $\mathcal{L}_c f(z) = \dfrac{1}{z}\left(\mathcal{L}f\right)\left(\frac{1}{z}\right)$. On remarque que ce changement de variable est une involution.

La transformée de Laplace admet un inverse fonctionnel nommé transformée de Borel. Cette transformée a aussi une formulation intégrale. Dans le cas de la transformée de Borel usuelle, on a $\mathcal{B}(f) = \dfrac{1}{2i\pi}\int_{c-i\infty}^{c+i\infty} \exp(zt)f(t)dt$ où c est plus grand que la partie réelle de toutes les singularités de $f(t)$.

De façon analogue, la transformée de Borel combinatoire est $\mathcal{B}_c(f) = \dfrac{1}{2i\pi}\int_{c-i\infty}^{c+i\infty} \dfrac{\exp(zt)}{t} f(1/t)dt$ où c est plus grand que la partie réelle de toutes les singularités de $f(1/t)/t$. Le lient avec la transformée de Borel usuelle vient du changement de variable $\mathcal{B}_c(f) = \mathcal{B}(1/zf(1/z))$ ou de manière équivalente $\mathcal{B}(f) = \mathcal{B}_c(f(1/z))' = \mathcal{B}_c(1/zf(1/z))$. Encore une fois, ce changement de variable est une involution.

Maintenant, concentrons nous sur les versions combinatoires de ces opérateurs. La transformée de Laplace combinatoire permet de faire le pont entre série génératrice exponentielle ($\sum_{n\geqslant 0} a_n \dfrac{z^n}{n!}$) et série génératrice ordinaire ($\sum_{n\geqslant 0} a_n z^n$). Plus précisément :

$$\mathcal{L}_c\left(\sum_{n\geqslant 0} a_n \frac{z^n}{n!}\right) = \sum_{n\geqslant 0} a_n z^n.$$

De manière réciproque, on a :

$$\mathcal{B}_c\left(\sum_{n\geqslant 0} a_n z^n\right) = \sum_{n\geqslant 0} a_n \frac{z^n}{n!}.$$

De ces formules sur les séries génératrices, on obtient facilement les identités suivantes :
— $\mathcal{L}_c f' = \frac{1}{z}(\mathcal{L}_c f - f_0)$,
— $\mathcal{L}_c(\int f) = z\mathcal{L}_c f$,
— $\mathcal{B}_c(zf) = \int \mathcal{B}_c f$,
— $\mathcal{B}_c(\frac{f-f_0}{z}) = (\mathcal{B}_c f)'$.

Comme pour la transformée de Laplace usuelle, le produit de transformées de Laplace combinatoires peut être exprimé à l'aide du produit de convolution :

$$z\mathcal{L}_c f \times \mathcal{L}_c g = \mathcal{L}_c(\int_0^z f(t)g(z-t)dt).$$

De manière équivalente :

$$\mathcal{L}_c f \times \mathcal{L}_c g = \mathcal{L}_c(\int_0^z f(t)g'(z-t)dt + g_0 f(z)).$$

Le produit ordonné (cf. chapitre 2) donne donc une interprétation combinatoire à cette convolution. On notera $f * g$ le produit de convolution combinatoire $\int_0^z f(t)g'(z-t)dt + g_0 f(z)$.

De même, le produit de transformées de Borel peut être exprimé à l'aide d'un produit de convolution dans le plan complexe :

$$\mathcal{B}f \times \mathcal{B}g = \mathcal{B}(\frac{1}{2i\pi}\int_{c-i\infty}^{c+i\infty} f(t)g(z-t)dt).$$

En composant cette dernière formule avec le changement de variable liant transformée de Borel usuelle et transformée de Borel combinatoire, on obtient :

$$\mathcal{B}_c f \times \mathcal{B}_c g = \mathcal{B}_c(\frac{1}{2i\pi}\int_{c-i\infty}^{c+i\infty} \frac{1}{(1-zt)t} f(1/t)g(z/(1-zt))dt).$$