Université de Caen Normandie
École doctorale MIIS

Habilitation à diriger des recherches

Spécialité : Informatique

défendue par
Julien David

Vers une compréhension théorique du comportement des algorithmes en pratique

soutenue le 4 Novembre 2024

Après avis des rapporteurs :
M. Arnaud Carayol ..... Prof., & Université Gustave Eiffel
M. Philippe Duchon ..... Prof., & Université de Bordeaux
M. Vlady Ravelomanana ..... Prof., & Université Paris Cité

Devant le jury composé de :
Mme. Marie-Pierre Béal ..... Prof., & Université Gustave Eiffel
M. Julien Clément ..... CR. HDR, Université de Caen

UNIVERSITÉ CAEN NORMANDIE

Je veux adresser de sincères remerciements à l'ensemble de l'équipe AMACC, qui m'a accueilli il y a 3 ans. J'y ai trouvé un climat de travail idyllique, précieux, sans lequel cette habilitation n'aurait sans doute pas vu le jour. Il m'est difficile de faire des remerciements spécifiques tant il règne dans cette équipe un climat de bienveillance et d'entraide, de sérieux professionnel sans tomber pour autant dans l'austérité ou l'élitisme. Qu'il s'agisse de discuter d'informatique théorique ou appliqué, de recherche, d'enseignement ou de politique du laboratoire, de trouver à la dernière minute quelqu'un pour me remplacer en TP à cause d'un imprévu, de trouver de l'aide pour réparer un vélo ou une voiture, de discuter d'un différent avec un collègue ou un étudiant ou simplement de faire une pause café, les membres de l'équipe sont toujours disponibles et j'espère être à la hauteur pour leur rendre la pareille quand iels en ont et en auront besoin. Que chacun et chacune soit assurée de ma sincère gratitude et pour ceux et celles qui le souhaite, de mon amitié. Je me dois tout de même de remercier en particulier à Loick Lhote, avec qui je collabore plus particulièrement. C'est un brillant pédagogue, un scientifique à mon humble avis sous-évalué, rigoureux, profondément humain. Nos échanges de ces 3 dernières années m'ont énormément appris et ont fortement contribué au contenu de ce manuscrit. Je me dois également, sous peine d'être chassé de notre bureau, de nommer Julien Courtiel, pour toutes les raisons précédemment mentionnées et sans qui je n'aurai sous doute personne avec qui partager ma cinéphilie. Je me dois, sous peine d'avoir une question vache lors de la soutenance, de nommer Julien Clément, qui a accepté d'être garant pour se manuscrit et dont l'enseignement des quatre accords toltèques est une source d'apaisement. Puissions nous d'ailleurs retrouver un moment pour jouer d'autres accords. Je me dois, sous peine de le voir disparaître à jamais après sa soutenance, de remercier Antonin Callard pour son amitié indéfectible. To Mostafa Gholami : forgive me for quoting your favorite poet in French. "Quand ton tour viendra, ne soupire pas — car chacun à son heure boira à la coupe fatale" (Omar Khayyam).

Mon intégration réussie au GREYC n'est pas que le fait des membres de l'équipe AMACC. Merci à Bruno Crémilleux qui, en me proposant ne nouer des collaborations avec des collègues de Rouen, via la fédération Normastic, m'a permis de réaliser un souhait inexprimé depuis des années : travailler avec Thierry Lecroq. L'article de Thierry Lecroq et Simone Faro sur la comparaison des algorithmes de recherches de facteurs dans une séquence nourrissait chez moi à la fois admiration et frustration : admiration pour le travail accompli, car nécessaire et fastidieux ; Frustration à cause des générateurs aléatoires utilisés. Le générateur aléatoire du chapitre 3 est né de mes réflexions autour de leur article. Le remerciement suivant va donc évidemment à Thierry Lecroq, Richard Groult et Bastien Auvray, avec qui travailler est un réel plaisir : de ceux où l'on se surprend à ne pas réussir à décrocher d'une preuve ou d'une expérience tant que celle ci ne sera pas terminée et où chacun apporte une contribution égale.

Je pense également à mes anciens collègues parisiens : Olivier Bodini, Mathieu Lacroix, Joseph Le Roux, Roland Grappe, Florian Sikora, Lucie Galand, Pierre Boudes, Flavien Breuvart, Pierre Rousselin, Mehdi Naïma ; tout comme à mes anciens et actuels collègues auvergnats et lyonnais : Arnaud Mary, Pierre Colomb, Yannick Loiseau, Laurent Beaudou, Lhouari Nourine.

Merci à mes trois rapporteurs (Philippe Duchon, Arnaud Carayol, Vlady Ravelomanana) pour leur relecture rigoureuse, qui a considérablement amélioré la qualité du manuscrit. Merci à Marie-Pierre Béal pour avoir accepté de faire partie de mon jury, malgré le fait qu'elle soit très sollicitée.

Che io sia sempre accompagnato dalla moderazione, dove sorge il sol dell'avvenir.

1

Introduction

Selon Donald E. Knuth [Don11], un *algorithme* est un ensemble de règles permettant d'obtenir une sortie spécifique à partir d'une entrée spécifique.

Les algorithmes apparaissent dès l'antiquité :

- que ce soit chez les Babyloniens il y a 3800 ans, pour résoudre des problèmes de géométrie (retrouver les dimensions d'une citerne) ou d'économie (le capital et les intérêts d'un prêt après une période donnée).
- en Grèce antique, avec le célèbre algorithme d'Euclide (il y a 2300 ans) permettant de calculer le plus grand commun diviseur de deux nombres.

Le mot lui-même est dérivé du nom du savant perse al-Khwârizmî, qui donna au 9e siècle une classification des algorithmes existant.

L'apparition de l'informatique au 20e siècle va permettre à l'algorithmique de prendre son essor. Pour citer Knuth une fois de plus, un programme informatique est l'énoncé d'un algorithme dans un langage bien défini.

Rapidement, deux courants se développent pour déterminer l'algorithme le plus efficace pour résoudre un problème : l'approche expérimentale, qui consiste à exécuter des programmes sur un même jeu de données et à mesurer les ressources consommées ; l'approche théorique qui consiste à étudier la complexité des algorithmes.

1 Analyse d'algorithmes

L'analyse d'un algorithme consiste à estimer la quantité de ressources consommées lors de son exécution. Le plus souvent, les ressources auxquelles on s'intéresse sont le temps et l'espace mémoire.

Soit $\mathbb{O}$ l'ensemble des objets combinatoires qui constituent l'entrée de l'algorithme. On nomme taille d'un objet combinatoire $o \in \mathcal{O}$, notée $|o|$, le nombre d'atomes qui le constitue : les lettres d'un mot, les noeuds d'un arbre, les maillons d'une liste chaînée. On note $\mathbb{O}_n$ l'ensemble des objets de taille n, c'est à dire

$$\mathbb{O}_n = \{o \in \mathbb{O} \mid |o| = n\}$$

Pour n fixé, $\mathbb{O}_n$ est un ensemble fini. Le coût d'un algorithme est une fonction $cout : \mathcal{O} \to \mathbb{N}$ qui a chaque objet combinatoire associe la quantité de ressources utilisées.

On nomme *pire cas d'un algorithme*, qu'on notera p_n, le coût maximal de l'algorithme parmi l'ensemble $\mathbb{O}_n$ des entrées de même taille, c'est à dire $p_n = max\{cout(o) \mid o \in \mathbb{O}_n\}$.

La complexité d'un algorithme est l'estimation de l'ordre de grandeur du coût de l'algorithme lorsque sa taille tend vers l'infini. La complexité dans le pire des cas d'un algorithme est donc l'étude de $\lim_{x\to\infty} p_n$. De façon similaire on peut définir la complexité dans le meilleur des cas m_n en considérant le coût minimal de l'algorithme. Enfin, pour une distribution de probabilité donnée π sur $\mathbb{O}_n$, il est possible de définir le coût moyen :

$$\mu_n = \sum_{o\in\mathbb{O}_n} cout(o) \times \pi(o)$$

où $\pi(o)$ est la probabilité d'obtenir l'objet o. Si on note $\mathbb{O}_{n,k}$ l'ensemble des objets de taille n pour lesquels l'algorithme a un coût k, on a :

$$\pi(\mathbb{O}_{n,k}) = \sum_{o\in\mathbb{O}_{n,k}} \pi(o) \text{ et } \mu_n = \sum_{k=m_n}^{p_n} k\pi(\mathbb{O}_{n,k})$$

Les constantes multiplicatives et additives étant de moindre importante face à l'ordre de grandeur, la notation de Landau permet de simplifier l'expression de la complexité. Ainsi, dans ce manuscrit, on retrouvera les notations suivantes. Pour tout $x_n \in \{p_n, m_n, \mu_n\}$, on note

- $x_n = \mathcal{O}(f(n))$ si
$$\exists c \in \mathbb{R}^+, \limsup_{x\to\infty} \frac{x_n}{f(n)} \leq c$$
- $x_n = \Omega(f(n))$ si
$$\exists c \in \mathbb{R}^+, \liminf_{x\to\infty} \frac{x_n}{f(n)} \geq c$$
- $x_n = \Theta(f(n))$ si
$$x_n = \mathcal{O}(f(n)) \text{ et } x_n = \Omega(f(n))$$
- $x_n = o(f(n))$ si
$$\exists c \in \mathbb{R}^+, \lim_{x\to\infty} \frac{x_n}{f(n)} = 0$$

L'analyse dans le pire des cas est la plus répandue et reste la méthode principale utilisée pour hiérarchiser l'efficacité des algorithmes permettant de résoudre un même problème. Cette méthode montre pourtant ses limites car l'ensemble

des entrées qui réalisent le pire des cas peuvent ne jamais se produire en pratique. La hiérarchie obtenue via la complexité dans le pire des cas peut alors se révéler trompeuse comme dans les exemples suivants :

- Dans le problème de la recherche d'un mot de taille m dans un texte de de taille n (voir Chapitre 6), l'algorithme naïf ($p_n = \Theta(nm)$) **peut** se révéler plus rapide en pratique que l'algorithme de Knuth-Morris-Pratt [KMP77] ($p_n = \Theta(n)$).
- Les algorithmes de calcul des traverses minimales d'un hypergraphe (voir Chapitre 5), l'algorithme de Fredman et Khachiyan [FK96], dont la complexité dans le pire des cas est quasi-polynomiale dans la taille de l'entrée et de la sortie, est connu pour être bien moins efficace, en pratique, que l'algorithme de Berge [Ber89] dont la complexité dans le pire des cas peut être exponentielle dans la taille de l'entrée et de la sortie.
- il est possible d'effectuer un prétraitement sur les entrées du tri rapide [Hoa61] ($p_n = \Theta(n^2)$), avec lequel l'algorithme sera en pratique aussi rapide (voire plus rapide) que celui du tri fusion ($p_n = \Theta(n \log n)$).

Le "risque" de se limiter à une analyse dans le pire des cas est donc de produire une hiérarchie de l'efficacité des algorithmes qui soit erronée en pratique.

L'analyse du coût moyen μ_n d'un algorithme, ou analyse en moyenne, vient donc s'ajouter, en complément, à l'analyse dans le pire des cas. On sait par exemple que

- Pour la distribution uniforme sur les mots de taille m dans les textes de taille n, l'algorithme naïf a une complexité moyenne $m_n = \Theta(n)$, ce qui explique donc pourquoi il **peut** être efficace face à l'algorithme de Knuth-Morris-Pratt.
- Pour la distribution uniforme sur les permutations de taille n, l'algorithme du tri rapide a un coût moyen de $\mu_n = \Theta(n \log n)$.

On voit bien ici en quoi l'analyse en moyenne a permis de mieux comprendre ce qui était observé de façon empirique à l'aide de résultats théoriques. Toutefois, les détracteurs de l'analyse en moyenne lui reprochent l'utilisation de la distribution uniforme, arguant que celle ci n'est pas représentative de contextes réels. Reprenons les exemples précédents afin de vérifier cet argument.

- dans la recherche de séquence dans du texte, l'analyse en moyenne de l'algorithme naïf prévoit que le nombre de comparaisons effectuées est inférieur à $c_k n$ où c_k est une constante qui dépend de la taille de l'alphabet sur lequel les mots sont définis. De plus $c_k < 2$, pour tout $k \geq 2$. En pratique cette tendance est vérifiée.
- pour le calcul des traverses minimales d'un hypergraphe, si l'on considère la distribution uniforme sur les hypergraphes à n sommets, alors avec une probabilité qui tend vers 1, les hypergraphes contiennent un nombre exponentiel d'hyperarêtes et l'algorithme de Berge est polynomial dans la taille de l'entrée uniquement. En pratique, les hypergraphes contiennent bien moins d'hyperarêtes et le comportement de l'algorithme de Berge est donc radicalement différent.
- dans le cas de l'algorithme du tri rapide, le pré-traitement évoqué consiste précisément à mélanger l'entrée en temps $\Theta(n)$, afin de garantir que, quel que soit le contexte de départ, l'algorithme se comportera comme l'analyse en moyenne le prévoit.

On voit donc avec ces exemples, trois cas de figure différents pour l'analyse en moyenne selon la distribution uniforme. Dans le premier cas, l'analyse en moyenne prévoit le comportement de l'algorithme, mais sans que l'on comprenne pourquoi (du moins à ce stade du récit) celui-ci est similaire à celui observé en pratique, alors que les données d'entrées ne se ressemblent clairement pas. Dans le second cas, l'analyse en moyenne n'est pas pertinente. Dans le dernier, tout comme dans le second cas, l'analyse en moyenne ne permet pas de prédire l'efficacité de l'algorithme dans tous les contextes pratiques. En revanche, l'analyse en moyenne a permis de produire un prétraitement, qui perturbe la donnée d'entrée de façon à retrouver la distribution uniforme.

On peut ajouter à cela d'autres exemples qui donnent raison aux détracteurs, comme par exemple [KNR19], qui montrent qu'une expression régulière engendrée selon la distribution uniforme parmi les expressions de même taille peut presque sûrement être simplifiée en une expression de taille constante. De même l'automate minimal associé à un automate non-déterministe de taille arbitrairement grande, engendré selon la distribution uniforme, possède presque sûrement un nombre constant d'états [Cha+04]. Autrement dit, pour certaines classes d'objets, la distribution uniforme produit presque sûrement des objets qui manquent d'intérêt.

Depuis les deux dernières décennies, de nombreux travaux ont donc tenté de faire de l'analyse d'algorithmes de façon différente, afin de combler les manques de l'analyse dans le pire des cas et de l'analyse en moyenne pour la distribution uniforme.

Dans [Val+09], les auteurs montrent que si l'entrée de l'algorithme est produite par une source dynamique aux bonnes propriétés, alors la complexité moyenne du tri rapide dépend de **l'entropie de Shannon de la source**. Plus l'entropie est élevée, plus on se rapproche de la distribution uniforme et d'un comportement moyen en $\Theta(n \log n)$. Plus l'entropie diminue et plus on s'en éloigne pour se rapprocher du pire des cas. Ils montrent également que l'entropie de Shannon et l'entropie de collision interviennent dans la complexité moyenne des algorithmes du tri insertion et du tri à Bulle. Dans [BDN12], nous avons montré que la complexité moyenne de l'algorithme de Moore [Moo56] de minimisation des automates déterministes est en $\Theta(n \log n)$ quelle que soit la distribution de probabilité sur la structure de transitions sous-jacente de l'automate, tant que la distribution de probabilité sur les états terminaux suivait une loi de Bernoulli. Dans [MNW14] et [ANP16], les auteurs considèrent différents modèles théoriques pour tenter de prévoir de nombre moyen d'erreurs de prédictions de branchement effectuées par le processeur (à chaque test effectué par un algorithme, le processeur tente de prévoir la réponse afin de précharger les instructions suivantes. Les erreurs de

prédictions peuvent avoir un coût non négligeable) pour différents algorithmes comme le tri rapide, quickSelect, ou une méthode d'exponentiation. Dans [ANP16], les auteurs parviennent même à améliorer l'efficacité de quickSelect et de l'exponentiation en réduisant le nombre d'erreurs de prédictions.

Mes travaux de recherche s'inscrivent dans cette dynamique : analyser des algorithmes afin d'obtenir des résultats théoriques qui correspondent au mieux à des phénomènes observés en pratique. Je m'intéresse donc à la complexité dans le pire des cas, mais aussi celle en moyenne, selon une distribution uniforme ou non. Le contexte et la difficulté d'analyse fixeront la méthode utilisée. Une approche semble toutefois se dessiner. Imaginons que l'on possède une fonction $CoutMoyen$ qui prend en entrée une distribution π_n sur les entrées de taille n d'un algorithme et renvoie le nombre moyen d'instructions effectué par l'algorithme selon cette distribution. Considérons maintenant un ensemble D_n de distributions de probabilité sur les entrées de taille n d'un algorithme. On s'intéresse à l'intégrale suivante :

$$\int_{\pi_n \in D_n} CoutMoyen(\pi_n)\, d\mu(\pi_n)$$

Autrement dit, on veut calculer une moyenne de complexité moyenne. On développera cette idée dans un interlude avant les chapitres 5 et 6, dans lesquelles on retrouvera ladite méthode, avec des approches différentes. Il convient maintenant de déterminer quel ensemble D_n pourrait être intéressant à étudier.

Dans le chapitre 5, l'un des modèles probabilistes considéré reprend cette idée d'une intégrale, où toutes les distributions sur $]0,1[^n$ sont considérées, mais pondérées via une mesure μ.

Dans le chapitre 6, on considère un ensemble fini $D_{k,t}$ de distributions de dimension k partageant toutes la même entropie de Shannon t, ou la même entropie de collision t (ou une valeur proche de t). On considère ensuite la distribution uniforme sur les éléments de $D_{k,t}$. On obtient ainsi des résultats sur la complexité moyenne de l'algorithme naïf de recherche de séquence pour ces familles de distributions.

2 Génération aléatoire d'objets combinatoires

L'analyse d'algorithmes est un travail difficile. Il peut être compliqué de se forger une intuition sur le comportement moyen d'un algorithme, ou encore de réaliser quelles sont les propriétés sur les entrées qui modifient le comportement de l'algorithme.

Un générateur aléatoire est un algorithme qui produit des objets combinatoires selon une distribution de probabilité fixée au préalable. Ces générateurs sont des outils précieux pour les chercheurs en analyse d'algorithmes. Ils permettent d'évaluer expérimentalement le comportement des algorithmes à étudier et d'émettre des conjectures sur leur complexité moyenne, pour une distribution donnée. Ils permettent également d'évaluer les statistiques des objets engendrés. Certaines de ces propriétés peuvent être une clé de compréhension du comportement moyen d'un algorithme. Par exemple, dans [FN16], les auteurs montrent que, asymptotiquement presque sûrement, un automate déterministe aléatoire contient un grand nombre de grands cycles disjoints, accessibles depuis l'état initial. Ce résultat leur permet de montrer que l'algorithme de Brzozowski est presque sûrement super-polynomial.

Afin d'analyser des algorithmes ou des objets combinatoires en moyenne, il est donc possible d'utiliser la méthodologie suivante :

1. **Simulation** : on utilise un générateur pour produire des objets aléatoirement. On effectue des mesures sur les objets produits (coût d'un algorithme, valeur d'une propriété).
2. **Conjecture** : En observant l'évolution des mesures produites selon la taille de l'objet engendré, on émet des conjectures sur les comportement asymptotiques.
3. **Preuve** : on utilise les intuitions obtenues grâce aux conjectures pour obtenir une preuve sur le comportement moyen de l'algorithme, ou les propriétés moyennes de l'objet étudié.

Cette méthode n'a bien sûr rien d'obligatoire et de nombreuses personnes passent directement à la troisième étape sans passer par les deux premières.

Plusieurs méthodes génériques permettent de décrire des générateurs aléatoires d'objets combinatoires. Certaines méthodes se basent sur les séries génératrices associées aux objets combinatoires à engendrer. La méthode classique pour obtenir ces séries se nomme "Méthode Symbolique" [FS09] : on part d'une spécification combinatoire de l'objet, puis à l'aide d'un dictionnaire de séries génératrices, on transforme la spécification en série. Une fois la série obtenue, on peut en extraire les coefficient et les stocker. En reprenant la spécification combinatoire et en la couplant avec les informations obtenues sur les différents coefficients, il est possible d'obtenir un générateur aléatoire : c'est la méthode récursive [FZV94]. Les coefficients précalculés et stockés étant rapidement très grands, la taille des objets que l'on peut engendrer avec cette méthode est limitée. Une autre méthode évite ces précalculs en relâchant une contrainte : celle d'engendrer des objets de taille fixe : on considère que tout objet γ d'une classe $\mathcal{C}$, de série génératrice $C(x)$ a une probabilité $\frac{x^{|\gamma|}}{C(x)}$ d'être engendré, avec x une valeur inférieure au rayon de convergence de C. C'est la méthode de Boltzmann [Duc+04]. La difficulté consiste alors à choisir le paramètre x de telle sorte que la taille moyenne des

objets engendrés soit la taille que l'on désire, c'est à dire $x\frac{C'(x)}{C(x)} = n$. Pour obtenir un objet de taille donnée, on utilise alors la *méthode du rejet* : tant que l'objet engendré ne possède pas la taille souhaitée, on le jette et on recommence. L'efficacité de la méthode dépend donc de la distribution de la taille de l'objet en sortie, ce qui va bien sûr dépendre de la classe combinatoire et de la série génératrice associée. La méthode de Boltzmann a été utilisée pour obtenir de nombreux générateurs sur des objets variés tels que : des partitions planes [BFP10], des graphes planaires [Fus09], des instances de métamodèles [Mou+09], des expressions booléennes [Gar05], des automates déterministes accessibles [BN07], des lambda termes [BGJ13], des polynominos convexes [Bod+13a], . . . Ce manuscrit ne contient pas de générateurs aléatoires basés sur l'une de ces deux méthodes, mais elles sont suffisamment importantes pour que l'on en fasse mention.

Lorsqu'il semble impossible d'obtenir une spécification combinatoire de l'objet à engendrer, il est possible de construire des générateurs aléatoires à l'aide de chaînes de Markov. Cette partie sera détaillée dans un interlude, juste avant le chapitre 3.

3 Plan du manuscrit

Une partie de ma recherche est donc centrée sur de l'algorithmique, en particulier de l'analyse d'algorithmes. Une autre partie de ma recherche va donc consister à produire des générateurs aléatoires de structures combinatoires.

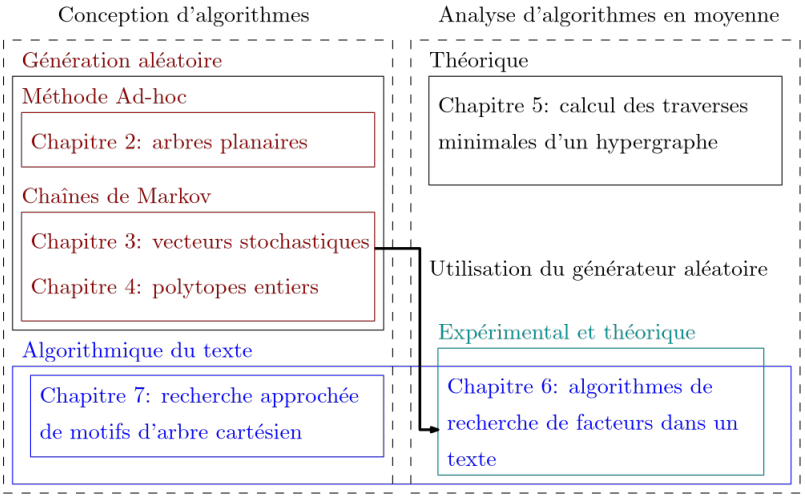


FIG. 1.1 : Plan du manuscrit

Le chapitre 2 présente un algorithme de génération aléatoire d'arbres planaires où la distribution des degrés des sommets est fixée, en un nombre quasi-optimal de bits aléatoires. Un premier interlude fournira quelques explications sur les chaînes de Markov, afin de faciliter la compréhension des chapitres suivants. Le chapitre 3 présente un générateur aléatoire de vecteurs stochastiques possédant des propriétés données. L'idée derrière ce résultat est de pouvoir engendrer aléatoirement. . . des générateurs aléatoires. En effet, nous avons mentionné que ce manuscrit contient de l'analyse d'algorithmes, où l'on considère un ensemble $D_{k,t}$ de distribution de probabilités de dimension k possédant une propriété commune t, comme une même entropie de Shannon ou une même entropie de collision. Afin d'estimer expérimentalement la valeur de l'intégrale mentionnée, il serait donc intéressant de posséder un générateur aléatoire d'éléments de $D_{k,t}$. Le chapitre 4 présente un générateur aléatoire de polytopes entiers compris dans un hypercube. Ces deux derniers chapitres utilisant des techniques à base de chaînes de Markov, il a été nécessaire de décrire des graphes dont l'ensemble des sommets est l'ensemble des structures combinatoires que l'on souhaite engendrer aléatoirement, puis d'étudier la connexité de ces graphes. Le chapitre 4 contient donc une étude de la connexité des graphes de polytopes.

Dans la seconde partie, je présenterai mes résultats en algorithmique. Après un second interlude où j'exprime un point de vue sur l'analyse d'algorithmes et où j'introduis une méthodologie pour l'analyse en moyenne qui ne se limite pas à la distribution uniforme, je tenterai dans les deux chapitres suivants d'appliquer cette méthodologie dans deux contextes distincts. Le chapitre 5 présente des résultats obtenus sur l'analyse des algorithmes de calcul des traverses minimales d'un hypergraphe. Le chapitre 6 utilise le générateur aléatoire obtenu dans le Chapitre 3 pour analyser expérimentalement et théoriquement les algorithmes de recherche de facteurs dans du texte. On verra que, comme pour le cas du tri rapide, l'entropie joue un rôle fondamental dans le comportement de ces algorithmes. Le chapitre 7 reste dans la thématique de la recherche de motifs dans des séquences. La notion de motif considérée est basée sur celle des arbres cartésiens. On présentera une version approchée de cette recherche de motif, c'est à dire qu'on autorisera à ce qu'une zone du texte soit légèrement différente du motif considéré. On fournira bien sûr une analyse des algorithmes obtenus.

La conclusion présente mon projet de recherche à court et moyen terme.

2

Arbres planaires et nombre de bits aléatoires quasi-optimal

Article(s) associé(s). *Olivier Bodini, Julien David and Philippe Marchal. Random-bit optimal uniform sampling for rooted planar trees with given sequence of degrees and Applications. CALDAM 2016 : 97-114.*

L'apport de ce chapitre porte non pas sur une amélioration de la complexité en temps et en espace de la génération aléatoire d'arbres planaires, mais sur la quantité de bits aléatoires utilisée pour effectuer le calcul. Souvent négligée car perçue comme une opération à coût constant, la génération d'un nombre aléatoire a, en pratique, un coût en temps non négligeable. S'il s'agit bel et bien d'un coût constant (sauf lors d'un appel à des bibliothèques particulières pour la génération de grands nombres), celui ci est moins négligeable qu'une simple suite d'opérations arithmétiques et réduire le nombre d'appels à ces fonctions permet généralement de diminuer de façon non négligeable le temps d'exécution.

Les arbres sont des objets combinatoires parmi les plus étudiés dans le domaine de la génération aléatoire et pour lesquels il existe de nombreuses méthodes efficaces. L'algorithme de Rémy [Rém85] est une méthode ad-hoc utilisant des propriétés combinatoires sur les arbres binaires. L'idée est de construire un arbre binaire à $2n-1$ noeuds dont les feuilles sont étiquetées de 1 à n, puis d'élaguer les feuilles pour ne garder que la partie non étiquetée de l'arbre. L'algorithme est linéaire en temps et en espace mais utilise $\Theta(n \log n)$ bits aléatoires, car chaque étape de l'algorithme consiste à sélectionner un noeud existant aléatoirement. D'autres algorithmes utilisent le principe de l'algorithme de Rémy pour engendrer d'autres familles d'arbres planaires [ARS97a ; ARS97b ; BBJ13]. Il est également possible d'utiliser des méthodes classiques de génération aléatoire pour engendrer des familles d'arbres planaires : la méthode récursive [FZV94 ; DPT10], la méthode de Boltzmann [Duc+04 ; BP10], les processus de Galton-Watson [Dev86]. La génération d'arbres dont le nombre de sommets ayant un degré donné est fixé a déjà été traitée dans [ARS97a]. Cependant le nombre de bits aléatoires utilisés n'est pas optimal [KY76]. Dans ce chapitre, on décrit deux méthodes pour la génération aléatoire d'arbres planaires dont la séquence des degrés des sommets est fixée en amont. La première est une redescription de [ARS97a] qui utilise la notion de mot de Lukasiewicz. La seconde méthode remplace les deux premières étapes de la première afin d'obtenir un générateur plus efficace en terme de bits aléatoires.

1 Définitions

On commence par définir la notion d'alphabet d'arbre. Ici chaque lettre correspond à un type de noeuds dans les familles d'arbres que l'on souhaite engendrer.

Définition 2.1 *Alphabet d'arbre*

Un alphabet d'arbre Σ_f est un couple (Σ, f) constitué d'un alphabet $\Sigma = (a_1, \dots, a_k)$ et d'une fonction $f : \Sigma : \mathbb{N} \cup \{-1\}$ qui associe à chaque symbole de Σ un entier, tel que :

- $f(a_1) = -1$
- $\forall 1 \leq i < k$, on a $f(a_i) \leq f(a_{i+1})$.

Autrement dit, les symboles de Σ sont ordonnés en fonction de leur image par f.

Dans cette définition, chaque lettre de l'alphabet a correspond à un type de noeud de l'arbre et $f(a)$ correspond au nombre d'enfants que possèdent les noeuds de ce type, décrémenté de 1. On décrit à présent les mots de Lukasiewicz, qui utilisent la notion d'alphabet d'arbre.

Définition 2.2 *Mots de Lukasiewicz et mots valides*

Un mot w défini sur un alphabet d'arbre $\Sigma_f = ((a_1, \dots, a_k), f)$ est un mot de f-Lukasiewicz si :

$$\forall i < k, \sum_{j=1}^{i} f(a_j) \geq 0 \tag{2.1}$$

$$\sum_{i=1}^{k} |w|_{a_i} f(a_i) = -1 \tag{2.2}$$

si seule la deuxième condition est respectée, on dit que w est valide.

Une autre façon de voir ces alphabets d'arbre est d'imaginer une marche dans le demi-plan. Chaque lettre a de l'alphabet correspond à un pas et $f(a)$ à la variation de la hauteur de la marche lors de ce pas. La Fig. 2.1 illustre cette bijection. Ainsi, si l'on considère l'alphabet $\Sigma = \{a, b\}$, avec $f(a) = -1$ et $f(b) = 1$, les mots de Lukasiewicz sont

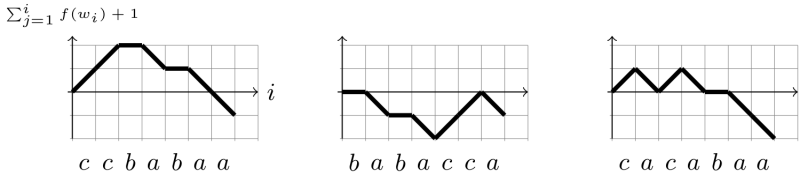


FIG. 2.1 : On suppose ici que l'alphabet d'arbre est $\Sigma = \{a, b, c\}$ tel que $f(a) = -1$, $f(b) = 0$ et $f(c) = 1$. Les trois exemples que l'on retrouve dans cette figure montrent des mots définis sur Σ et les marches dans le demi-plan associées. Seul le premier mot est un mot de Lukasiewicz. Le second est un mot valide. Le troisième mot ne respecte aucune des deux conditions.

exactement les mots de Dyck (auquel on ajoute un dernier 'a'), la marche dans le demi-plan associée les chemins de Dyck, dont on sait qu'ils sont en bijection avec les arbres binaires.

Pour tout alphabet d'arbre donné, l'ensemble des mots de Lukasiewicz définis sur cet alphabet est en bijection avec une famille d'arbres planaires. La bijection est relativement simple à calculer. On lit un mot u en même temps que l'on effectue un parcours préfixe de l'arbre associé. Pour chaque lettre u_i, si $f(u_i) = -1$, le noeud associé est une feuille et on passe au noeud suivant dans le parcours. Si $f(u_i) \geq 0$, le noeud associé est de degré $f(u_i) + 1$.

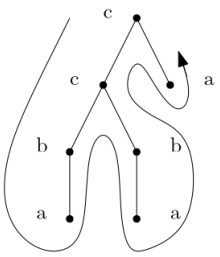


FIG. 2.2 : Arbre planaire associé au mot de Lukasiewicz *ccbabaa* de la FIG. 2.1.

2 Deux algorithmes de génération aléatoire

Les étapes des deux algorithmes sont représentés dans la FIG. 2.3. Comme on peut le voir, dans les deux cas, l'enjeu est d'engendrer aléatoirement non pas des mots de Lukasiewicz, mais des mots valides. La FIG. 2.4 illustre la transformation d'un mot valide de longueur n en mot de Lukasiewicz, faisable en temps $\Theta(n)$ (on considère que la taille de l'alphabet est constante).

On se concentre ici sur la méthode dichotomique, qui permet de réduire le nombre de bits aléatoires.

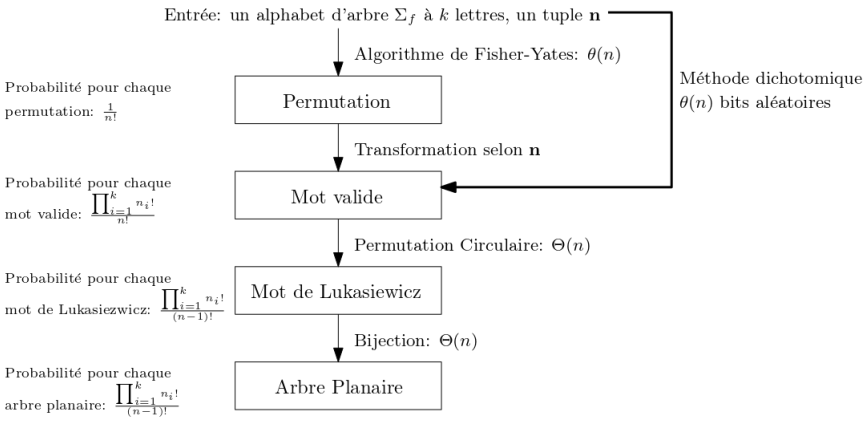


FIG. 2.3 : Cette figure résume le fonctionnement des deux algorithmes. Le premier utilise l'algorithme de Fisher-Yates pour engendrer une permutation aléatoire qui sera ensuite convertie en mot valide, alors que la seconde méthode engendre directement un mot valide en utilisant un nombre presque optimal de bits aléatoires. Les mots valides sont ensuite transformés en mot de Lukasiewicz grâce à une permutation circulaire. Il suffit ensuite d'utiliser la bijection pour transformer le mot de Lukasiewicz en arbre planaire.

2.1 Méthode Dichotomique pour engendrer un mot valide

Algorithme 1 : Mot valide aléatoire

Entrées : Un alphabet d'arbre Σ_f à k lettres, un tuple $\mathbf{n}$ dont les valeurs somment à n
Sorties : un mot valide w
$w \leftarrow \varepsilon$;
pour $i \in \{1, \cdots, n\}$ **faire**
  $a \leftarrow$ Tirer une lettre aléatoire selon $\mathbf{n}$;
  $w \leftarrow w \cdot a$;
  $\mathbf{n} \leftarrow$ mettre à jour la distribution en enlevant une occurrence de a ;
retourner w

L'algorithme 1 a une complexité linéaire et appelle n fois l'algorithme 2. Le nombre moyen de bits aléatoires tirés par le second algorithme est inférieur ou égal à $2 + \log_2(k)$, où k est une constante. Le générateur ainsi obtenu est optimal (à une constante multiplicative près) en nombre de bits aléatoires.

Algorithme 2 : Lettre aléatoire selon un vecteur $\mathbf{n}$

Entrées : Un alphabet d'arbre Σ_f à k lettres, un tuple $\mathbf{n} = (n_1, \ldots, n_k)$ dont les valeurs somment à n
Sorties : une lettre aléatoire a
$i \leftarrow 1$;
$j \leftarrow k$;
$min \leftarrow 0$;
$max \leftarrow n$;
tant que $i \neq j$ **faire**
  si $BitAleatoire() == 1$ **alors**
    $tmp \leftarrow min$;
    $min \leftarrow \frac{min+max}{2}$;
    tant que $min > (tmp + n_i)$ **faire**
      $i \leftarrow i + 1$;
      $tmp \leftarrow tmp + n_i$;
  sinon
    $tmp \leftarrow max$;
    $max \leftarrow \frac{min+max}{2}$;
    tant que $max < (tmp - n_j)$ **faire**
      $j \leftarrow j - 1$;
      $tmp \leftarrow tmp - n_j$;
retourner *la i-ème lettre de l'alphabet* Σ_f

2.2 Passage d'un mot valide à un mot de Lukasiewicz

Cette partie se base sur le lemme cyclique : parmi les n permutations circulaires d'un mot valide, il existe exactement un mot de Lukasiewicz. Ainsi, si l'on possède un générateur aléatoire uniforme de mots valides, que l'on transforme ce dernier en mot de Lukasiewicz, on obtient un générateur aléatoire uniforme de mot de Lukasiewicz.

3 De l'expérimentation aux résultats théoriques : un exemple sur les arbres de Motzkin

Nous avons ensuite mis en application la méthodologie "simulation, conjecture, preuve" ainsi que l'étude de l'évolution d'une propriété moyenne d'un objet combinatoire lorsqu'un paramètre évolue.

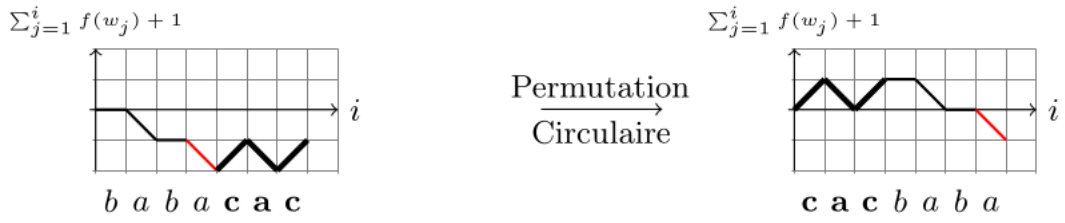


FIG. 2.4 : Le mot valide $babacac$ n'est pas un mot de Lukasiewicz, mais le mot $cacbaba$ en est un. L'idée pour passer de l'un à l'autre est de trouver le plus petit i tel que $\sum_{j=i}^{i} f(w_j)$ est minimal, puis de calculer le mot $w_{i+1} \cdots w_n w_1 \cdots w_i$.

Simulation À l'aide du générateur aléatoire obtenu, nous avons observé expérimentalement la hauteur des arbres unaires-binaires, dits de Motzkin, lorsque la proportion de noeuds unaires fluctue.

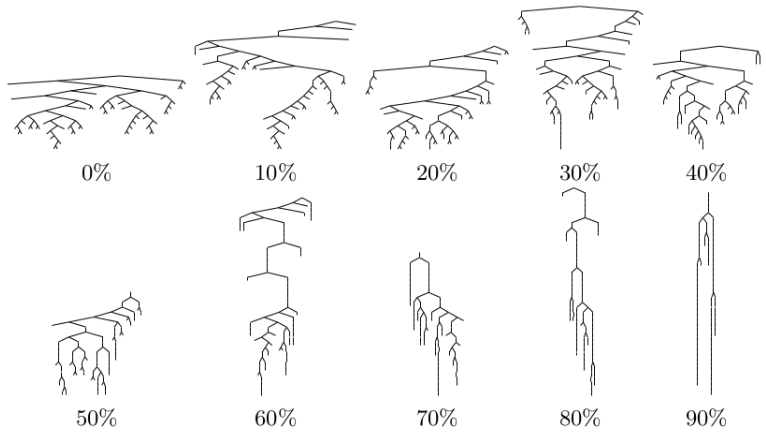


FIG. 2.5 : Exemples d'arbres de Motzkin engendrés aléatoirement en faisant varier la proportion de noeuds unaires.

La FIG. 2.5 montre les différentes formes que peuvent prendre des arbres de Motzkin, lorsque le pourcentage de noeuds unaire varie. La FIG. 2.6 montre l'évolution de la hauteur lorsque ce même pourcentage varie.

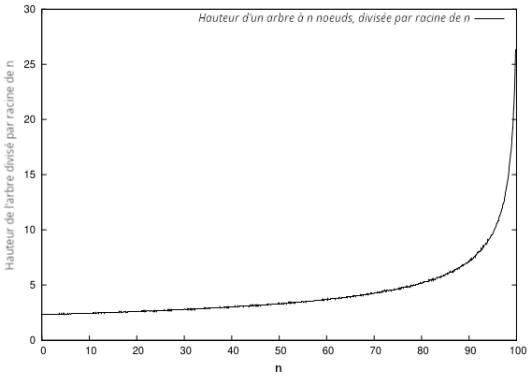


FIG. 2.6 : Cette figure montre l'évolution de la hauteur d'un arbre de Motzkin, divisée par la racine du nombre de noeuds, lorsque le pourcentage de noeuds unaires varie de 0 à 99, 9. Le nombre de noeuds de l'arbre est fixé à $n = 1000$. Pour chaque pourcentage testé, $10,000$ arbres ont été engendrés.

Conjecture Comme on peut le voir sur la FIG. 2.6, la hauteur des arbres de Motzkin varie entre $\sqrt{n}$ (hauteur moyenne des arbres binaires) et $n-1$ (hauteur d'une arbre ne contenant que des noeuds unaires). En affichant la hauteur, multipliée par $\frac{\sqrt{c_n}}{n}$ (où c_n désigne le nombre de feuilles moins 1), la courbe obtenue a permis de faire le lien avec les CRT. Un CRT (pour *Continuum Random Tree*) est un arbre aléatoire continu défini par Aldous [Ald93], fortement lié aux mouvements browniens. En particulier, la hauteur d'un CRT possède la même loi de probabilité que la hauteur maximale d'une

excursion brownienne. On obtient ainsi le résultat suivant.

Théorème 2.3 *Bodini-D.-Marchal*

Soit une suite d'entiers $(c_n, n \geq 1)$ telle que $c_n = o(n)$ et $(\log n)^2 = o(c_n)$. Pour la distribution uniforme sur les arbres de Motzkin à n sommets et $c_n + 1$ feuilles, on a presque sûrement

$$\frac{\sqrt{c_n}}{n} hauteur(T_n) \to H$$

où H est une variable aléatoire dont la loi de probabilité est celle de la hauteur d'un CRT.

À noter que ce résultat a été démontré, sous une forme plus générale, dans [BM14], chose que nous ignorions à l'époque.

4 Conclusion

Si ce travail s'inscrit dans la thématique de la génération aléatoire d'objets combinatoires, l'utilisation d'une méthode ad-hoc, basée sur de la combinatoire bijective, est unique dans mes travaux. En effet, au fur et à mesure que les objets se complexifient, on préférera l'utilisation de méthodes puissantes et génériques. Parmi celles ci, les générateurs aléatoires basés sur des chaînes de Markov, s'ils sont les moins efficaces, permettent de décrire des générateurs aléatoires quasi-uniformes, ou asymptotiquement uniformes, sur des ensembles d'objets que l'on sait mal décomposer récursivement.

Interlude : les chaînes de Markov

Un **processus stochastique**, ou processus aléatoire, est une famille

$$\{X_t\}_{t\in T}$$

de variables aléatoires X_t, où $t \in T$ représente le temps. On s'intéressera uniquement aux processus à **temps discret**, c'est-à-dire pour lesquels $T = \mathbb{N}$. Les valeurs que peuvent prendre ces variables X_t sont nommées **états** du processus et l'ensemble des valeurs possibles est appelé **espace des états**, que l'on notera A dans cet interlude. On considérera ici et dans le reste du manuscrit que l'espace des états est fini. Une **marche aléatoire** est un processus stochastique dans lequel :

- on se déplace dans l'espace en choisissant une "direction" aléatoirement à chaque instant t.
- la variable aléatoire X_t représente l'état où l'on se trouve à l'instant t.

Un processus stochastique possède la **propriété de Markov** si :

$$\forall n \in \mathbb{N}, \forall (i_1, \ldots, i_n, j) \in A^{n+1}, \mathbb{P}(X_{n+1} = j \mid X_0 = i_0, X_1 = i_1, \ldots, X_n = i_n) = \mathbb{P}(X_{n+1} = j \mid X_n = i_n)$$

Autrement dit, la probabilité d'être dans un état au temps $n+1$ ne dépend que de l'état où l'on se trouve au temps n.

Définition 2.4 *Chaîne de Markov*

Une **chaîne de Markov** est un :
- processus stochastique à temps discret
- possédant la propriété de Markov.

Les chaînes de Markov ont de nombreuses applications dans divers domaines : en physique statistique, en théorie de l'information, en bioinformatique ou encore dans la conception des moteurs de recherches. Nous les utiliserons ici pour concevoir des générateurs aléatoires.

Une chaîne de Markov est dite **homogène** si la probabilité de passer d'un état à un autre est indépendante du temps.

$$\forall n \in \mathbb{N} \setminus \{0\}, \forall i, j \in A, \mathbb{P}(X_{n+1} = j \mid X_n = i) = \mathbb{P}(X_n = j \mid X_{n-1} = i)$$

Dans une chaîne de Markov homogène, on a juste besoin de connaître :

- la distribution initiale π_0 :
  c'est à dire la probabilité d'être dans chaque état au temps 0.
- pour tout couple $(i, j) \in A^2$, la probabilité de passer de i à j (qui ne varie pas avec le temps). On peut représenter toutes ces probabilités dans une matrice stochastique M de dimensions $|A| \times |A|$, que l'on nomme **matrice de transition**.

Dans cet interlude et les chapitres qui suivent, les chaînes de Markov considérées sont toutes homogènes. Aussi, on omettra de le préciser. Une fois la distribution initiale π_0 et la matrice de transition M connues, il est possible de calculer les distributions $\pi_t, \forall t \in \mathbb{N}$, c'est à dire, pour chaque instant t, de connaître la probabilité d'être dans chaque état de A. Dans le cas général, la distribution au temps t est donc :

$$\forall t \in \mathbb{N}, \pi_t = \pi_0 \times M^t$$

Une distribution est dite **stationnaire** si $\pi M = \pi$.

Cette notion de distribution stationnaire va nous être utile par la suite. On souhaite décrire des générateurs aléatoires d'objets combinatoires et utilisant la notion de chaîne de Markov. Ce que la notion de distribution stationnaire nous apprend est que, si on effectue une marche aléatoire "suffisamment longue", et que l'on renvoie le dernier état de cette marche, on a alors engendré aléatoirement un élément de A suivant une distribution proche de la distribution stationnaire, ou égale à celle-ci.

Définition 2.5 *Irréductible*

Une chaîne de Markov homogène de matrice de transition M est **irréductible**. Pour tous états $x, y \in A$, il existe un temps $t \in \mathbb{N}$ tel que :

$$M^t(x, y) > 0$$

Autrement dit, le graphe sous-jacent est fortement connexe.

Définition 2.6 *Apériodique*

Soit une chaîne de Markov homogène de matrice de transition M. On définit l'ensemble des temps qui permettent de partir d'un état x et d'y revenir.

$$\mathcal{T}_x = \{t \geq 1 \mid M^t(x,x) > 0\}$$

La période de x est égal au **pgcd** de $\mathcal{T}_x$. Une chaîne est **apériodique** si l'ensemble de ses états a une période de 1. Autrement dit, le pgcd des longueurs des cycles vaut 1.

Une chaîne de Markov homogène de matrice de transition M possède une **unique** distribution stationnaire si elle est irréductible et apériodique. Reste maintenant à déterminer ce que vaut cette distribution stationnaire. Dans les chapitres suivants, on s'intéressera uniquement à la distribution uniforme sur A. On veut donc pouvoir caractériser les matrices de transition dont on peut garantir que leur distribution stationnaire est la distribution uniforme.

Définition 2.7 *Réversible*

Une chaîne de Markov de matrice de transition M est **réversible** si pour tout $x, y \in A$, on a

$$M(x,y) = M(y,x)$$

ou, autrement dit

$$\mathbb{P}(X_t = y \mid X_{t-1} = x) = \mathbb{P}(X_t = x \mid X_{t-1} = y)$$

la probabilité d'aller d'un état X à un état Y est égale à la probabilité d'aller de Y à X

Propriété 2.1

Si une chaîne de Markov homogène de matrice de transition M est
- irréductible,
- apériodique,
- réversible.

alors elle converge vers une unique distribution stationnaire : la distribution uniforme.

Pour construire un générateur aléatoire d'objets combinatoires qui engendre des objets de A selon la distribution uniforme on va donc, dans les chapitres suivants, décrire un ensemble de règles de transitions puis démontrer que la matrice ainsi obtenue est irréductible, apériodique et réversible.

La notion de **temps de mélange** sur les chaînes de Markov permet de connaître un temps à partir duquel on est "suffisamment proche" de la distribution uniforme. Cette notion permet de déterminer la complexité algorithmique de ces générateurs aléatoires.

La *distance en variation totale* permet de comparer les distributions de probabilité. Soit μ et ν deux distributions de probabilités sur un espace de probabilité Ω. On note $\|\mu - \nu\|_{TV}$ la distance en variation totale entre μ et ν, définie comme suit :

$$\|\mu - \nu\|_{TV} = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|$$

Soit π la distribution stationnaire d'une chaîne de Markov de matrice de transitions P. Le temps de mélange, noté $t_{mix}(\varepsilon)$ est égal à

$$t_{mix}(\varepsilon) = min\{t \mid \forall \mu, \|\mu P^t - \pi\|_{TV} \leq \varepsilon\}$$

Autrement dit, il s'agit du premier instant t à partir duquel, quelle que soit la distribution initiale μ, la distribution μP^t a une distance en variation totale à la distribution stationnaire inférieure à ε.

Les chapitres suivants ne contiennent pas de résultats précis sur le temps de mélange des chaînes de Markov. En revanche, on y retrouvera des études du diamètre des graphes sous-jacents, car cela constitue une borne inférieure significative du temps de mélange.

Propriété 2.2 *Diamètre et temps de mélange([LPW08], section 7.1.2, page 89)*

Soit δ le diamètre du graphe associé à une matrice de transition. Pour tout $0 < \varepsilon < \frac{1}{2}$, on a

$$t_{mix}(\varepsilon) \geq \frac{\delta}{2}$$

Notons ici que la borne inférieure en $\frac{\delta}{2}$ est en partie due au fait que ε peut avoir une valeur élevée (proche de $\frac{1}{2}$).

Nos algorithmes démarrent souvent d'un état initial (ou d'une distribution quelconque sur un sous-ensemble de l'espace des états). Dans la distribution π_0, il existe donc des états dont la probabilité associée est de 0. Comme on sait que la distribution stationnaire est l'uniforme, toutes les probabilités y sont non nulles. Il convient donc de borner le temps nécessaire pour pouvoir atteindre n'importe quel état de A. On utilisera donc en pratique une marche aléatoire de longueur au moins égale à δ et non à $\frac{\delta}{2}$.

D'autres méthodes, plus sophistiquées, se concentrent sur les plus grandes valeurs propres de la matrice de transition (voire sur le trou spectral de celle-ci), afin d'obtenir des informations sur le temps de mélange. Ces méthodes ne sont toutefois applicables que lorsque la matrice possède de bonnes propriétés, pour lesquelles il est plus facile d'obtenir des informations sur les valeurs propres. Aussi, pour les chaînes de Markov appliquées à la génération aléatoire d'objets combinatoires, on se contente généralement de bornes inférieures obtenues à l'aide du diamètre du graphe. En pratique, lorsque l'on effectuera des expériences, on calibrera les générateurs en utilisant la méthode suivante : si, malgré le fait que le nombre d'expérience sur lequel une moyenne est calculée est jugé suffisamment élevé (par exemple $10,000$ entrées), le résultat obtenu ne semble pas "stable", c'est à dire qu'il varie "trop" lorque l'expérience est répétée, on augmente le nombre de pas effectué par la chaîne de Markov, jusqu'à obtenir une stabilisation. Si cette méthode est bien sûr largement discutable d'un point de vue théorique (la variance de la variable aléatoire observée peut-être elle-même très élevée. Il peut exister des goulots d'étranglement dans le graphe sous-jacent à la chaîne de Markov, on obtient alors des résultats stables tout en étant très loin de la distribution stationnaire), elle semble en pratique suffisante dans les cas où j'ai été amené à les appliquer.

3

Vecteurs sources à propriété fixée

Article(s) associé(s). • *Julien David, Random Generation of Source Vectors with a fixed determining property. Soumis à TCS.*

Dans l'introduction, j'ai mentionné mon intention de faire de l'analyse d'algorithmes en considérant non pas une distribution de probabilité sur les entrées d'une taille n donnée, mais un ensemble de distributions D_n. On s'intéresse en particulier à l'intégrale

$$\int_{\pi_n \in D_n} \mathbb{P}(\pi) CoutMoyen(\pi_n)\, d\pi_n$$

Dans ce chapitre, on va se ramener à un cas où D_n est fini et où $\mathbb{P}(\pi) = \frac{1}{|D_n|}$. Plus précisément, dans ce chapitre, nous allons nous intéresser à des distributions π sur des alphabets de taille k, à partir desquelles on peut déduire des distributions sur des mots de taille n en voyant ces derniers comme étant des suites de n variables i.i.d tirés selon π. Les distributions considérées sur les mots de taille n donc des distributions produits notées $\pi_{\otimes n}$. L'analyse en moyenne d'un algorithme revient alors à étudier

$$\frac{1}{|D_k|} \sum_{\pi \in D_k} CoutMoyen(\pi_{\otimes n})$$

Nous avons également rappelé que les générateurs aléatoires sont des outils importants pour l'étude du comportement moyen des algorithmes. Dans cette perspective, on va donc décrire ici un générateur aléatoire d'éléments de D_k, soit un générateur aléatoire de distributions de probabilités sur les entrées d'un algorithme.

Une distribution de probabilités sur k évènements peut être encodée par un vecteur stochastique de dimension k. L'objectif de ce chapitre est d'engendrer aléatoirement et uniformément des vecteurs stochastiques discrétisés ayant des propriétés fixés.

On propose un algorithme pour engendrer uniformément une généralisation des vecteurs stochastiques, appelés **vecteurs sources**, dont l'image par une fonction donnée est fixée (on précisera le problème dans la section Définition).

Définition 3.1 *Vecteur Source*

Soit $s \leq 1$, un **vecteur source** de dimension k est une séquence de valeurs $(p_1, \ldots, p_k)$ telles que

$$\sum_{i=1}^{k} p_i = s.$$

Cette généralisation est nécessaire pour décrire l'algorithme, mais la finalité est de n'engendrer que des vecteurs sources dont la somme s vaut 1, c'est-à-dire des distributions de probabilités.

Le constat d'origine est le suivant : l'entropie d'une source semble avoir une incidence sur la complexité des algorithmes. Dans [Val+09], les auteurs montrent que la complexité moyenne de QuickSelect et QuickSort, lorsque les clés sont des mots, dépend de l'entropie de la source qui a engendré lesdits mots. De même, dire que les algorithmes de recherches de séquences dépendent de l'entropie de la source qui les engendrent relève presque du lieu commun. Une question qui n'a pourtant pas été étudiée est : comment l'entropie d'une source influence t-elle la complexité moyenne des algorithmes de recherche de séquences ? Nous étudierons cette question dans le chapitre 6, mais dans un premier temps, j'ai donc souhaité construire un générateur aléatoire de vecteurs stochastiques dont l'entropie de Shannon était fixée, afin de pouvoir réaliser des expériences sur des algorithmes de recherche de séquence.

Un vecteur source tiré uniformément a une entropie de Shannon moyenne proche du maximum [GS93] avec une variance qui tend vers 0. Il n'est donc pas envisageable d'utiliser une méthode à base de rejet pour obtenir des vecteurs sources avec une entropie donnée.

J'en suis donc arrivé à une méthode à base de chaîne de Markov. Les résultats présentés dans ce chapitre étant généralisables à d'autres fonctions, j'ai fini par élargir de problème à la génération aléatoire de vecteurs sources dont l'image par certaines fonctions particulières est fixée.

1 Définitions

Dans ce chapitre, on décrit un algorithme à base de Chaîne de Markov permettant d'engendrer aléatoirement des vecteurs sources. La preuve de l'irréductibilité de la chaîne a nécessité que le graphe sous-jacent soit fini, raison pour laquelle nous allons utiliser une version discrétisée du problème. On considère qu'une valeur x_i est un multiple d'une valeur minimum ε. On peut ainsi considérer que ε est une puissance négative de 2, ce qui est tout à fait cohérent avec la représentation des nombres sur un ordinateur.

Définition 3.2 *Ensemble de vecteurs sources*

Soient $0 < s \leq 1$, une dimension k et une valeur minimum ε, l'ensemble fini des vecteurs sources que l'on considère est

$$\mathbb{S}_{k,\varepsilon}(s) = \{(x_1, \ldots, x_k) \in (\varepsilon\mathbb{N}^{\geq 1})^{\,k} \mid \sum_{i=1}^{k} x_i = s\}$$

On notera simplement $\mathbb{S}_{k,\varepsilon}$ lorsque $s = 1$.

On dira d'une fonction qu'elle est unimodale s'il n'existe qu'un unique maximum local : le maximum global. On veut se restreindre à un sous-ensemble de $\mathbb{S}_{k,\varepsilon}(s)$ dont l'image par une fonction est égale à une valeur t.

- Soit $\oplus \in \{+, \times\}$ un opérateur associatif et commutatif et $\ominus \in \{-,\ /\}$ son opération inverse.
- Soit $H_k : \mathbb{R} \mapsto \mathbb{R}$ une injection,
- Soit $G :]0, 1] \mapsto \mathbb{R}_+ \setminus \{0\}$ une fonction continue.
- Soit $T_k : \mathbb{S}_{k,\varepsilon}(s) \mapsto \mathbb{R}$ une fonction **symétrique, unimodale et concave** telle que :

$$\forall (x_1, \ldots, x_k) \in \mathbb{S}_{k,\varepsilon}(s), T_k(x_1, \ldots, x_k) = H_k\left(G(x_1) \oplus \ldots \oplus G(x_k)\right)$$

La définition de T_k peut paraître assez restrictive. La propriété de symétrie est une conséquence de la définition, mais la concavité et l'unimodalité sont des restrictions. On donne quelques exemples de fonctions classiques qui rentrent dans le cadre de notre étude. Notons qu'il est possible d'adapter les algorithmes qui vont suivre pour gérer des fonctions convexes. Il est également possible de gérer certaines fonctions qui ne sont pas unimodales. Mais on perdrait alors l'aspect générique de la méthode.

Définition 3.3 *Entropie de Shannon*

L'entropie de Shannon $T_k(\pi)$ est définie comme

$$T_k(\pi) = \sum_{i=1}^{k} \pi_i \cdot \log_2(\frac{1}{\pi_i})$$

On a $0 \leq T_k(\pi) \leq \log_2(k)$.

- $\oplus = +$ et $\ominus = -$,
- $G(x) = -x\log_2(x)$
- $H_k(y) = y$ et $H_k^{-1}(z) = z$.

Définition 3.4 *Entropie de collision*

L'entropie de collision $T_k(\pi)$ est définie comme

$$T_k(\pi) = -\log_2(\sum_{i=1}^{k} \pi_i^2)$$

On a $0 \leq T_k(\pi) \leq \log_2(k)$.

- $\oplus = +$ et $\ominus = -$,
- $G(x) = x^2$
- $H_k(y) = -\log_2(y)$ et $H_k^{-1}(z) = 2^{-z}$.

Définition 3.5 *Produit*

Le produit $T_k(\pi)$ est défini comme

$$T_k(\pi) = \prod_{i=1}^{k} \pi_i$$

- $\oplus = \times$ et $\ominus = /$,
- $G(x) = x$
- $H_k(y) = y$ et $H_k^{-1}(z) = z$.

Définition 3.6 *Moyenne géométrique*

La moyenne géométrique $T_k(\pi)$ est définie comme

$$T_k(\pi) = \left(\prod_{i=1}^{k} \pi_i\right)^{\frac{1}{k}}$$

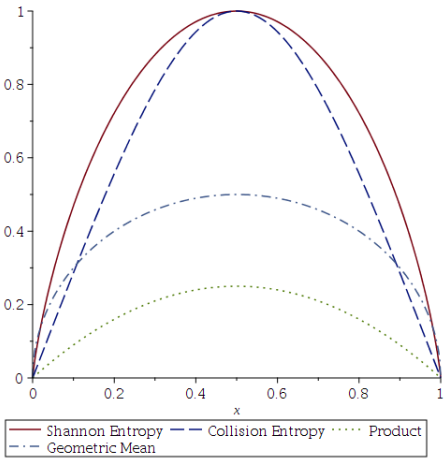
- $\oplus = \times$ et $\ominus = /$,
- $G(x) = x$
- $H_k(y) = y^{\frac{1}{k}}$ et $H_k^{-1}(z) = z^k$.

On voudrait à présent définir l'ensemble des solutions, c'est à dire l'ensemble des vecteurs sources que l'on souhaite engendrer aléatoirement et uniformément.

Définition 3.7 *Ensemble (naïf, mais pédagogique) de solutions*

Pour un k, ε, t fixé, on souhaite engendrer aléatoirement et uniformément un élément de

$$\{(x_1, \ldots, x_k) \in \mathbb{S}_{k,\varepsilon}(s) \mid T_k(x_1, \ldots, x_k) = t\}$$



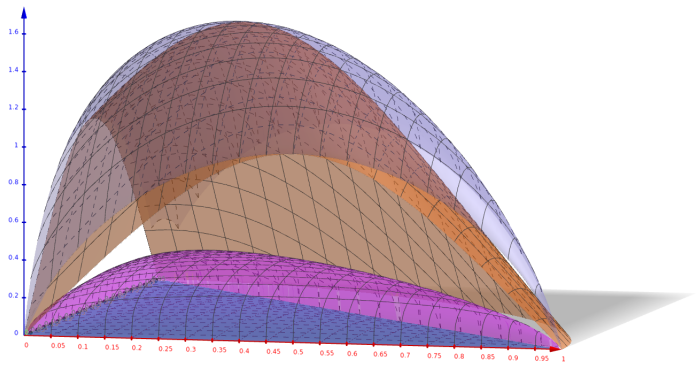


FIG. 3.1 : Les 4 fonctions présentées ci-dessus en dimensions 2 et 3 sont symétriques, concaves et unimodales. Leur valeur maximale en dimension k se situe toujours au point $(\frac{1}{k}, \dots, \frac{1}{k})$ et les valeurs minimales sont situées aux permutations de $(1-(k-1)\varepsilon, \varepsilon, \dots, \varepsilon)$.

Si l'on considère uniquement les multiples de ε, alors l'ensemble des vecteurs sources dont l'image par T_k est égale à t pourrait être vide. Ainsi, on définit l'ensemble des solutions qui approximent l'entropie visée. En dimension 2, l'ensemble des solutions $\mathbb{S}_{2,\varepsilon,t}(s)$ est un ensemble des vecteurs sources dont la distance avec la cible visée est minimale, c'est à dire :

$$\mathbb{S}_{2,\varepsilon,t}(s) = \begin{cases} \emptyset, \text{ si } t > T_2(\frac{s}{2}, \frac{s}{2}) \text{ et } t < T_2(s-\varepsilon, \varepsilon) \\ \underset{(x_1,x_2)\in\mathbb{S}_{2,\varepsilon}(s)}{\arg\min} \; |T_2(x_1,x_2) - t|, \text{ tel que } T_2(x_1,x_2) \neq T_2(x_1-\varepsilon, x_2+\varepsilon) \end{cases}$$

L'idée de la première ligne de cette définition est d'exclure les entropies dont la valeur est trop petite ou trop grande. Pour la seconde ligne, l'idée est de prendre des solutions qui minimisent la distance avec la cible. Dans de rares cas, il est possible que deux solutions très proches aient la même image par T. Dans ce cas, on ne conserve que l'une d'entre elles.

Grâce à cette définition, tout comme si l'on considérait le cas continu, l'ensemble contient :

- aucune solution si $t > T_2(\frac{s}{2}, \frac{s}{2})$ et $t < T_2(s-\varepsilon, \varepsilon)$
- une solution si $t = T_2(\frac{s}{2}, \frac{s}{2})$
- deux solutions $(x, 1-x)$ et $(1-x, x)$ sinon.

En plus grandes dimensions, on voulait conserver cette jolie propriété. C'est pourquoi on considère un hyperplan qui est parallèle à $k-2$ axes, ce qui revient à fixer $k-2$ valeurs dans le vecteur source. Puis, on considère parmi ces vecteurs sources ceux dont la distance avec l'entropie visée est minimale. De nouveau, pour un plan donné, il y a soit 0, 1 ou 2 solutions.

Pour tout $k \geq 3$ et $2 \leq \ell < k$, on définit la fonction $Update_{k,\ell} : \mathbb{R} \times [0,1]^\ell \mapsto \mathbb{R}$ comme suit :

$$Update_{k,\ell}(z, x_1, \dots, x_\ell) = \begin{cases} 0, \text{ si } T_\ell(x_1, \dots, x_\ell) \geq z, \\ H_{k-\ell}(H_k^{-1}(z) \ominus G(x_1) \ominus \dots \ominus G(x_\ell)), \text{ sinon} \end{cases}$$

La fonction $Update$ permet de connaître la contribution que devront fournir les $k-\ell$ variables restantes, une fois que ℓ variables ont déjà été fixées.

Lemme 3.8

Pour $\ell < k$, soit $\{\{a_1, \dots, a_\ell\}, \{b_1, \dots, b_{k-\ell}\}\}$ une partition de $\{1, \dots, k\}$. Pour ε, t et s fixés, on a pour tout $(x_1, \dots, x_k) \in \mathbb{S}_{k,\varepsilon}(s)$:

$$T_k(x_1, \dots, x_k) = t \iff T_{k-\ell}(x_{b_1}, \dots, x_{b_{k-\ell}}) = Update_{k,\ell}(t, x_{a_1}, \dots, x_{a_\ell})$$

La fonction Update nous permet à présent de définir l'ensemble des solutions recherchées, une fois que $k-2$ variables ont été fixées.

Définition 3.9 *Ensemble de solutions*

$$\begin{aligned}\mathbb{S}_{k,\varepsilon,t}(s) = \{(x_1,\dots,x_k) \in \mathbb{S}_{k,\varepsilon}(s) \mid & \exists i<j<k, \exists s' \in \varepsilon\mathbb{N},\\ & s' = \sum_{\ell\in\{1,\dots,k\}\setminus\{i,j\}} x_\ell,\\ & t' = Update_{k,\ell}(z, x_1,\dots,x_{i-1},x_{i+1},\dots,x_{j-1},x_{j+1},\dots,x_k)\\ & (x_i,x_j) \in \mathbb{S}_{2,\varepsilon,t'}(s-s')\}\end{aligned}$$

On veut donc obtenir un générateur aléatoire uniforme pour $\mathbb{S}_{k,\varepsilon,t}$, une version biaisée de la version continue.

Dans la suite de ce chapitre, on commence par décrire deux générateurs ad-hoc pour $\mathbb{S}_{2,\varepsilon,t}(s)$ et $\mathbb{S}_{3,\varepsilon,t}(s)$. Le générateur pour $k=3$ utilise celui pour $k=2$. Lorsque $k>3$, on décrit un générateur à base de chaîne de Markov pour $\mathbb{S}_{k,\varepsilon,t}$. Chaque étape de la marche aléatoire est en réalité un appel au générateur ad-hoc pour $k=3$.

2 Quelques propriétés

Lemme 3.10

Soient $(x_1,\dots,x_k) \in \mathbb{S}_{k,\varepsilon}(s)$ et les valeurs $k\in\mathbb{N}$, ε et $s\le 1$. Pour tout vecteur $(y_1,\dots,y_\ell)$, $\ell\le k$, tel que $\sum_i^\ell y_i = \sum_i^\ell x_i$, on a

$$T_k(y_1,\dots,y_\ell,x_{\ell+1},\dots,x_k) = T_k(x_1,\dots,x_k) \iff T_\ell(y_1,\dots,y_\ell) = T_\ell(x_1,\dots,x_\ell)$$

Autrement dit, il est possible de passer une solution de $\mathbb{S}_{k,\varepsilon,t}(s)$ à une autre, en modifiant exactement ℓ variables $(x_1,\dots,x_\ell)$. Pour cela, il est nécessaire et suffisant que les variables obtenues $(y_1,\dots,y_\ell)$ aient la même image par T_ℓ et la même somme. Pour effectuer une marche aléatoire sur les solutions de $\mathbb{S}_{k,\varepsilon,t}(s)$ il est donc possible d'utiliser un générateur aléatoire en dimension inférieure.

Lemme 3.11

Pour tout k, $s\ge 1$ et tout $t\in[T_k(s-(k-1)\varepsilon,\varepsilon,\dots,\varepsilon),\dots,T_k(\frac{s}{k},\dots,\frac{s}{k})]$, il existe $x\in\varepsilon\mathbb{N}^{\ge 1}$ avec $\varepsilon\le x\le\frac{s}{k}$, tel qu'il existe $y\in\varepsilon\mathbb{N}^{\ge 1}$ avec $y<\frac{s-(k-2)x}{2}$ et

$$(\underbrace{x,\dots,x}_{k-2},y,s-(k-2)x-y) \in \mathbb{S}_{k,\varepsilon,t}(s)$$

Ce lemme va nous permettre d'initialiser la chaîne de Markov en trouvant facilement un élément de $\mathbb{S}_{k,\varepsilon,t}(s)$. L'idée est qu'il est possible de considérer la même valeur x pour les $k-2$ premières dimensions. La symétrie des fonctions joue évidemment un rôle important dans la validité de ce Lemme. Le fait que la fonction soit unimodale nous permet de trouver x facilement.

2.1 Description de l'algorithme pour $k=2$

Lorsque $k=2$, si T_k est différentiable, il est possible d'utiliser la méthode de Newton pour trouver une solution de l'équation $T_2(x,1-x)=t$. Il suffit ensuite de renvoyer $(x,1-x)$ ou $(1-x,x)$ avec une probabilité $\frac{1}{2}$. Lorsque T_k n'est pas différentiable, il est possible d'effectuer une recherche dichotomique. L'algorithme 3.2 ci-dessous détaille le calcul dans le cas où la fonction est différentiable.

La recherche dichotomique est plus simple à programmer car elle ne nécessite pas de calculer la dérivée de T, mais elle converge moins rapidement vers la solution.

Algorithme 3 : $RandomSourceVectorK2(s,t)$

Entrées : s et t tels que $T_2(s-\varepsilon,\varepsilon) \leq t \leq T_2(\frac{s}{2},\frac{s}{2})$
Sorties : Un couple $(x, s-x) \in \mathbb{S}_{2,\varepsilon,t}(s)$
$x \leftarrow \frac{s}{4}$;
$\delta \leftarrow 1$;
$t' \leftarrow T_2(x, s-x)$;
tant que $t' \neq t$ *or* $\delta \geq \varepsilon$ **faire**
  $\delta \leftarrow \frac{t'}{T_2'(x,s-x)}$;
  $x \leftarrow x - \delta$;
  si $x < 0$ **alors**
    $x \leftarrow -x$;
  $t' \leftarrow T_2(x, s-x)$;
si $T_2(x-\varepsilon, s-x+\varepsilon) = T_2(x, s-x)$ **alors**
  $x \leftarrow x - \varepsilon$;
retourner $(x, s-x)$ *avec probabilité* $\frac{1}{2}$ *et* $(s-x, x)$ *sinon*

FIG. 3.2 : Algorithme pour $k = 2$

2.2 Description de l'algorithme pour $k = 3$

Pour $k = 3$, on définit un algorithme *ad-hoc* qui utilise le générateur de vecteur source pour $k = 2$. Une partie de l'algorithme 3.3 est laissée volontairement floue afin de préserver la lisibilité. L'idée est d'être capable de tirer une valeur $x' < s'$ et telle qu'il existe au moins un couple (y', z') avec $(x', y', z') \in \mathbb{S}_{3,\varepsilon,t'}(s')$. On discutera de cette question dans le paragraphe suivant. De plus, pour un x' donné, il est possible qu'il existe une ou deux solutions. Afin de préserver l'uniformité du générateur aléatoire, il est nécessaire de rejeter les cas où il n'existe qu'une solution avec probabilité $\frac{1}{2}$. L'algorithme étant le pas de base de la chaîne de Markov que nous souhaitons construire, il est possible de passer en entrée un vecteur de dimension supérieure à 3.

Algorithme 4 : RandomSourceVectorK3

Entrées : Un vecteur source π de dim. k
Sorties : Un vecteur source modifié π de dim. k
$(x, y, z) \leftarrow$ tirer 3 variables aléatoirement parmi k ;
$t \leftarrow T_3(x, y, z)$;
$s \leftarrow x + y + z$;
$x' \leftarrow$ Tirer aléatoirement et uniformément une valeur **valide** selon s et t ;
$s' = s - x'$;
$t' \leftarrow Update_{3,1}(t, x')$;
$(y', z') \leftarrow RandomSourceVectorK2(s', t')$;
si $y \neq z$ *ou avec probabilité* $\frac{1}{2}$ **alors**
  Remplacer (x, y, z) par (x', y', z') dans π ;
retourner π

FIG. 3.3 : Algorithme pour $k = 3$

Comment tirer une valeur valide x ? La FIG. 3.4 montre un exemple de valeurs minimum et maximum que l'entropie de Shannon peut prendre en dimension 3 en fonction de la valeur de la variable x. Notons que lorsque $t \geq T_2(\frac{s}{2},\frac{s}{2})$, l'intervalle de valeurs que x peut prendre est uniquement déterminé par l'entropie maximum. Aussi, lorsque $t < T_2(\frac{s}{2},\frac{s}{2})$, il existe deux intervalles de valeurs valides.

On note $maxInv_0(s,t)$ et $maxInv_1(s,t)$ les deux solutions de l'équation

$$t - T_3(x, \frac{s-x}{2}, \frac{s-x}{2}) = 0 \tag{3.1}$$

Ici $maxInv_0$ est la fonction inverse de l'équation 3.1 quand $x \in [\varepsilon, \frac{s}{3} - \varepsilon]$ et $maxInv_1$ quand $x \in [\frac{s}{3}, s - 2\varepsilon]$.
On note également $minInv_0(s,t)$ la plus petite solution de l'équation

$$t - T_3(x, s-x-\varepsilon, \varepsilon) = 0$$

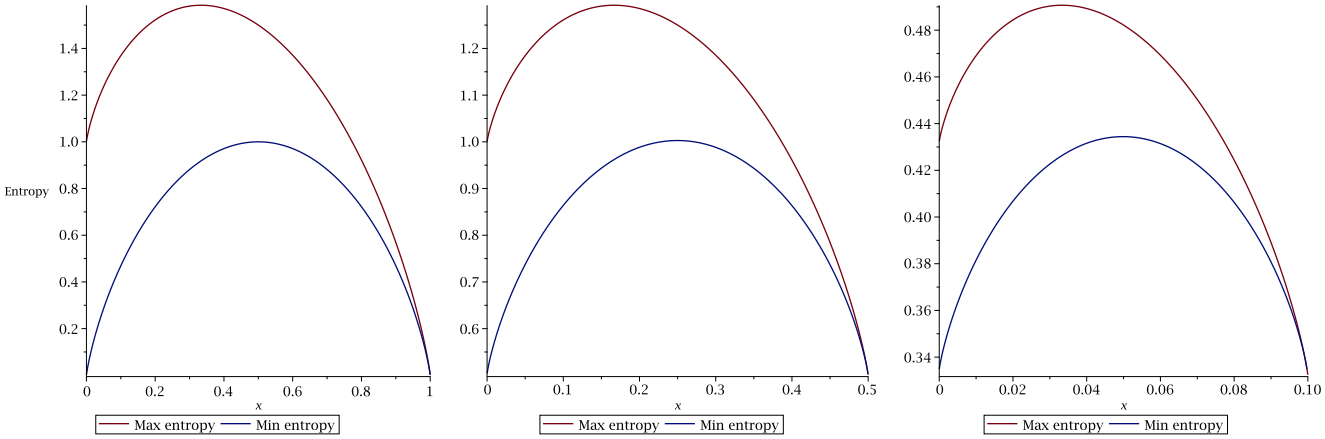


FIG. 3.4 : Cette figure représente les valeurs minimum et maximum que l'entropie de Shannon peut prendre, selon la valeur de x, pour $s = 1$, $s = 0.5$ et $s = 0.1$ (de gauche à droite). En fixant une entropie, c'est à dire en traçant une droite horizontale d'ordonnée t, la valeur que x est autorisée à prendre afin que le système ait une solution est située sur cette droite, au dessus de la courbe bleue et en dessous de la courbe rouge. Il s'agit d'un intervalle ou de l'union de deux intervalles.

Afin de simplifier la formule, quand $\oplus = +$ et ε est "suffisamment" proche de 0, on peut remplacer l'équation par $t - T_2(x, s - x) = 0$. Dans ce cas, il devient évident que l'autre solution de l'équation est $s - minInv_0(s,t)$ puisque T_k est symétrique. Quand $\oplus = \times$ on nommera $minInv_1(s,t)$ l'autre solution de l'équation.

Lemme 3.12

Soient $\oplus = +$, $t \in]0, \ldots, \log_2(3)]$ et $s \in]0, \ldots, 1]$, le système

$$\begin{cases} T_3(x,y,z) = t \\ x + y + z = s \end{cases}$$

possède une solution si et seulement si

$$x \in \begin{cases}]0, \ldots, minInv_0(s,t)] \cup [s - minInv_0(s,t), \ldots, maxInv_1(s,t)] \text{ si } t < T_2(\frac{s}{2}, \frac{s}{2}) \\ [maxInv_0(s,t), \ldots, maxInv_1(s,t)], \text{ sinon.} \end{cases}$$

Lemme 3.13

Soient $\oplus = \times$, $t \in]0, \ldots, \log_2(3)]$ et $s \in]0, \ldots, 1]$, le système

$$\begin{cases} T_3(x,y,z) = t \\ x + y + z = s \end{cases}$$

possède une solution si et seulement si

$$x \in \begin{cases}]0, \ldots, minInv_0(s,t)] \cup [minInv_1(s,t), \ldots, maxInv_1(s,t)] \text{ si } t < T_3(\frac{s}{2}, \frac{s-\varepsilon}{2}, \varepsilon) \\ [maxInv_0(s,t), \ldots, maxInv_1(s,t)], \text{ sinon.} \end{cases}$$

La FIG. 3.5 illustre le Lemme 3.12 dans le cas de l'entropie de Shannon. En pratique, les valeurs des fonctions inverses sont calculées avec la méthode de Newton ou une recherche dichotomique.

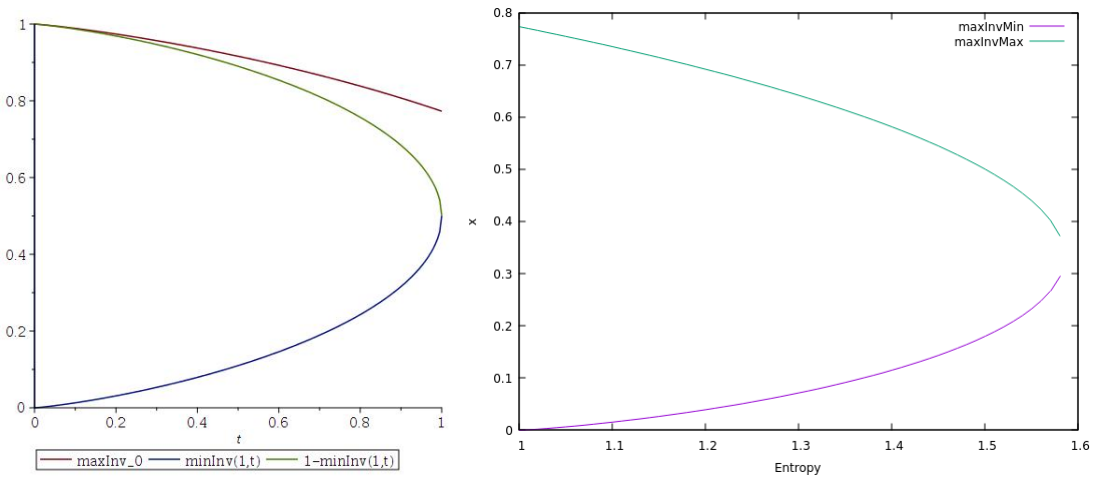


FIG. 3.5 : Ici, on considère que $s = 1$. La figure de gauche représente le cas où $t < T_2(\frac{s}{2}, \frac{s}{2})$. Pour un t fixé, les valeurs valides doivent être tirées soit sous la courbe bleue, soit entre la verte et la rouge. La figure de droite montre le cas où $t \geq T_2(\frac{s}{2}, \frac{s}{2})$.

2.2.1 ▶ Description de l'algorithme pour $k > 3$

La section qui suit décrit deux algorithmes qui sont essentiels pour créer un générateur aléatoire à base de chaînes de Markov : l'initialisation de la chaîne de Markov et le générateur aléatoire en lui-même.

Initialisation L'idée de l'initialisation est similaire à celle de l'algorithme $RandomSourceVectorK2(s, t)$. Soit une cible t sur un vecteur source de dimension k, on veut trouver un valeur x tel qu'il existe une valeur y et

$$(\underbrace{x, \ldots, x}_{k-2}, y, s - (k-2)x - y) \in \mathbb{S}_{k,\varepsilon,t}$$

Un tel y existe si

$$T_2(1 - (k-2)x - \varepsilon, \varepsilon) \leq Update_{k,k-2}(t, \underbrace{x, \ldots, x}_{k-2}) \leq T_2(\frac{1-(k-2)x}{2}, \frac{1-(k-2)x}{2})$$

ce qui, d'après les Lemmes 3.10 et 3.8, est équivalent à :

$$t \in [T_k(\underbrace{x, \ldots, x}_{k-2}, s - (k-2)x - \varepsilon, \varepsilon), \ldots, T_k(\underbrace{x, \ldots, x}_{k-2}, \frac{s-(k-2)x}{2}, \frac{s-(k-2)x}{2})]$$

L'algorithme 5 effectue donc une recherche dichotomique pour trouver une telle valeur x, dont l'existence est garantie par le Lemme 3.11.

Le générateur aléatoire L'algorithme 6 décrit la génération aléatoire des éléments de $\mathbb{S}_{k,\varepsilon,t}$.

Théorème 3.14

L'algorithme 6 converge vers la distribution stationnaire uniforme.

En effet, on démontre que la chaîne de Markov est :

- **irréductible** : l'idée de la preuve est que, pour toute distribution de départ π et toute distribution μ d'arrivée, il est possible d'atteindre une distribution ν depuis π à l'aide d'un unique appel à $RandomSourceVectorK3$ et telle que la distance totale de variation entre ν et μ est inférieure à la distance entre π et μ. Autrement dit, il est toujours possible de se rapprocher de la destination. L'espace des états étant fini, il existe donc toujours un chemin de π à μ
- **apériodique** : la chaîne étant irréductible, il suffit de montrer qu'il existe un cycle de taille 1 dans le graphe associé à la chaîne de Markov. Puisqu'il est possible que la fonction $RandomSourceVectorK3$ renvoie une erreur ou le même triplet que celui contenu dans la distribution de départ, tout état de la chaîne possède une boucle, ou cycle de taille 1.

Algorithme 5 : $EtatInitial(k,t)$

Entrées : Une fonction T_k, les valeurs s, k et t telles que $T_k(s-(k-1)\varepsilon,\dots,\varepsilon) \leq t \leq T_k(\frac{s}{k},\dots,\frac{s}{k})$
Sorties : Un vecteur source $\pi \in \mathbb{S}_{k,\varepsilon,t}(s)$
$x \leftarrow \frac{s}{k}$;
$\delta \leftarrow \frac{s}{2k}$;
$sum \leftarrow (k-2)x$;
tant que $t \notin [T_k(\underbrace{x,\dots,x}_{k-2}, s-sum-\varepsilon, \varepsilon), \dots, T_k(\underbrace{x,\dots,x}_{k-2}, \frac{s-sum}{2}, \frac{s-sum}{2})]$ **faire**
  si $t > T_k(\underbrace{x,\dots,x}_{k-2}, s-sum-\varepsilon, \varepsilon)$ **alors**
    $x \leftarrow x-\delta$;
  sinon
    $x \leftarrow x+\delta$;
  $\delta \leftarrow \frac{\delta}{2}$;
  $sum \leftarrow (k-2)x$;
$\delta_t \leftarrow Update_{k,k-2}(t, \underbrace{x,\dots,x}_{k-2})$;
$(y,z) \leftarrow RandomSourceVectorK2(s-sum, \delta_t)$;
retourner $(x,\dots,x,y,z)$

Algorithme 6 : Générateur aléatoire à base de chaîne de Markov

Entrées : Deux valeurs k et t telles que $T_k(1-(k-1)\varepsilon,\dots,\varepsilon) \leq t \leq \log_2(k)$, un nombre d'étapes $steps$
Sorties : Un vecteur source π de dimension k tel que $T_k(\pi) = t$
$\pi \leftarrow MarkovChainInitialState(k,t)$;
pour tous les $i \in \{1,\dots,steps\}$ **faire**
  $\pi \leftarrow RandomSourceVectorK3(\pi)$;
retourner π

- **réversible** : cette propriété est assez immédiate puisque la probabilité de passer d'une distribution à une autre dépend de la probabilité de tirer les 3 variables sur lesquelles les distributions diffèrent et de la probabilité d'être obtenu via un appel à $RandomSourceVectorK3$. Or cette fonction est elle-même un générateur aléatoire uniforme.

Le code source est disponible à l'adresse : *https ://git.unicaen.fr/julien.david/vectors*. Il permet d'engendrer des vecteurs sources en fixant les paramètres suivants : entropie de Shannon, entropie de collision, produit et moyenne géométrique. Il permet également d'engendrer aléatoirement des textes où chaque lettre est tirée de façon aléatoire et indépendante à l'aide d'une distribution issue du générateur précédent.

3 Perspective

La méthode présentée dans ce chapitre peut-être étendue. Il semble possible de se passer de la contrainte d'unimodalité : cela nécessite d'adapter les algorithmes de recherches de solution, mais ne devrait pas changer fondamentalement la chaîne de Markov. Il est également de l'adapter pour certaines fonctions convexes. En effet, ces fonctions sont maximales en $T_k(s-(k-1)\varepsilon, \varepsilon, \dots, \varepsilon)$. Il est donc possible de passer d'une fonction convexe à une fonction concave définit comme

$$U_k(x_1,\dots,x_k) = T_k(s-(k-1)\varepsilon,\varepsilon,\dots,\varepsilon) - T_k(x_1,\dots,x_k)$$

Si la fonction U_k possède les propriétés énoncés dans ce chapitre, alors les algorithmes sont encore valides. On pourrait ainsi imaginer engendrer des vecteurs sources dont la "distance" à la distribution uniforme est égale à une valeur t. On pourrait par exemple considérer la distance totale de variation [LPW08], ou la divergence de Kullback-Leibler [KL51]

Comme mentionné à la fin de la section précédente, ce générateur peut-être étendu pour faire de la génération aléatoire d'objets combinatoires divers. Pour un alphabet Σ à k lettres, muni d'une distribution $\pi \in \mathbb{S}_{k,\varepsilon,t}$, il est possible d'étendre la distribution aux mots de longueur n définis sur Σ en considérant la mesure produit $\mu_{\otimes\pi}$. Autrement dit, un mot est une séquence de variables aléatoires i.i.d. selon π. On considère ce générateur aléatoire pour faire de l'analyse d'algorithmes dans le chapitre 6.

Ou pourrait également imaginer adapter cette méthode aux arbres de Galton-Watson, où le type de chaque noeud est tiré selon une distribution $\pi \in \mathbb{S}_{k,\varepsilon,t}$ parmi les k types de noeuds existant.

A terme, j'espère également pouvoir décrire des générateurs aléatoires de graphes et d'hypergraphes, intéressants pour l'analyse d'algorithmes. Comme on le verra dans le chapitre 5, les produits de probabilités interviennent dans les statistiques des hypergraphes et le comportement moyen des algorithmes qui s'y appliquent. En revanche, il s'agit de modèles plus généraux, dans lesquels on s'intéresse certes à l'ensemble $\mathbb{S}_{k,\varepsilon,t}(s)$, mais où s peut-être égal à k (on considère alors des vecteurs de valeurs comprises entre 0 et 1).

4

Polytopes entiers

Article(s) associé(s). • *Julien David, Rado Rakotonarivo, and Lionel Pournin. Elementary moves on lattice polytopes. Journal of Combinatorial Theory, Series A 172, 105200 (2020).*
• *Julien David, Lionel Pournin, Rado Rakotonarivo, Random generation of lattice polytopes. Gascom 2018, 132–139 (2018).*
J'ai co-encadré la thèse de Rado Rakotonarivo, dirigée par Lionel Pournin, soutenue en mars 2020.

Les travaux décrits dans ce chapitre sont nées d'une volonté de collaboration avec Lionel Pournin, qui a mené au co-encadrement de la thèse de Rado Rakotonarivo. L'idée initiale était de faire de la génération aléatoire de polytopes entiers. J'apportais mes compétences en génération aléatoire et Lionel Pournin les siennes sur les polytopes. Nous avons rapidement réalisé que les opérations que nous définissions pour la génération aléatoire soulevaient bon nombre de questions sur les polytopes. La géométrie n'est vraiment pas mon domaine de prédilection. Dans ce chapitre, je vais donc tenter de donner brièvement des intuitions des résultats obtenus en me concentrant sur la dimension 2. Or, pour paraphraser Günter Ziegler, les intuitions dans les dimensions 2 et 3 ne fonctionnent pas en dimensions supérieures. Certains détails essentiels au bon fonctionnement des preuves seront donc omis, mais la lectrice ou le lecteur est invité(e) à consulter les articles mentionnés dans l'entête du chapitre, pour plus de détails. Certains théorèmes de l'article *Elementary moves on lattice polytopes* ont été obtenus alors que j'étais mis à disposition chez *Qwant*. N'y ayant pas contribué, je les ai omis de ce manuscrit.

Un *polytope* peut-être vu comme l'enveloppe convexe d'un ensemble fini de points dans $\mathbb{R}^d$. Si l'ensemble fini des points considérés est inclus dans $\mathbb{Z}^d$, on dit alors qu'il s'agit d'un *polytope entier*.

Ces polytopes apparaissent en optimisation combinatoire [Nad89 ; NZ11 ; Seb99], en géométrie discrète [BP92 ; BV97 ; LZ91] et sont en lien avec des variétés toriques [Bog+15 ; DDP09]. Notons que, dans le cas des polytopes entiers, l'alternance de suppression et d'ajout de sommets (que nous allons considérer dans ce chapitre) a été considérée, de sorte que le nombre de points entiers contenus dans le polytope change exactement de un [BS18 ; BS19 ; BGM16]. Ces modifications peuvent affecter la combinatoire des bornes d'un polytope de façon arbitraire, mais permet d'énumérer les polytopes entiers contenant un nombre fixé de points entiers [BS18 ; BS19]. Par contraste, nos opérations peuvent être utilisées pour énumérer les polytopes ayant un nombre fixé de sommets contenus dans un hypercube.

Un polytope est dit de *dimension pleine* si son intérieur est non-vide. Autrement dit, dans un espace à d dimensions, un polytope est en dimension pleine s'il est lui même de dimension d. Le nombre minimal de sommets requis pour être en dimension pleine dans un espace de dimensions d est $d+1$. Les polytopes en dimension pleine à $d+1$ sommets sont appelés les simplexes. En dimension 2 par exemple, ce sont les triangles. En dimension 3, il s'agit de pyramides à base triangulaire. Dans ce chapitre, on supposera que $d \geq 2$. Soient un polytope P et un hyperplan H tel que P est totalement contenu dans l'un des demi-espaces définis par H. L'intersection $F = P \cap H$ est appelée *face* du polytope P. Chaque face est elle-même un polytope. Si P est de dimension d et F de dimension $d-1$, F est alors une *facette* de P. Par exemple, dans un carré C, les sommets et les arêtes de C sont des faces. Les arêtes sont des facettes. Les facettes d'un cube sont elles-mêmes des carrés. Celles d'un simplexe de dimension 3 sont toutes des triangles.

Dans ce chapitre, on s'intéresse à la génération aléatoire de polytopes entiers en dimension pleine. Afin de considérer des objets combinatoires, c'est à dire des ensembles d'objets pour lesquels l'ensemble des objets d'une taille donnée est fini, on se limitera au final aux polytopes entiers inclus dans un hypercube $\{0, \ldots, k\}^d$ de dimension d et de côté k. Sauf mention contraire, le lecteur ou la lectrice pourra considérer que les polytopes sont en dimension pleine. Soit $x \in \mathbb{R}^d$ un point dans un espace en d dimensions. On notera x_i la i-ème coordonnée de x. Les travaux les plus proches de ceux présentés dans ce chapitre se concentrent sur la génération aléatoire de polyominos (généralisation des dominos) digitallement convexes [Bod+13b] et celle des polygones convexes à l'intérieur d'un disque [DDT14]. La méthode utilisée étant celle des chaîne de Markov, on commence par définir des opérations permettant de passer d'un polytope à un autre. On obtient ainsi un graphe de polytopes où les arêtes indiquent qu'il est possible de passer d'un objet à un autre en appliquant l'une des opérations. Le graphe obtenu nous servira ensuite à définir la matrice de transitions de la chaîne de Markov. On donnera enfin des résultats sur sa connexité et son diamètre, qui seront utiles pour déterminer si la chaîne est irréductible et pour obtenir une borne inférieure sur le temps de mélange.

Définition 4.1 *Insertion d'un sommet*

Soit un polytope P de dimension d, contenu dans $\mathbb{R}^d$ et $\mathcal{V}$ l'ensemble de ses sommets. Un sommet $x \in \mathbb{R}^d$ peut être inséré dans P si l'enveloppe convexe de $P \cup \{x\}$ admet $\mathcal{V} \cup \{x\}$ comme ensemble de sommets.

Définition 4.2 *Suppression d'un sommet*

Soit un polytope P de dimension d, contenu dans $\mathbb{R}^d$ et $\mathcal{V}$ l'ensemble de ses sommets. Un sommet $x \in \mathcal{V}$ peut être supprimé dans P si l'enveloppe convexe de $\mathcal{V} \setminus \{x\}$ reste d-dimensionnelle.

Ces opérations n'étant pas limitées aux polytopes entiers, il est possible de considérer différents graphes, en modifiant l'ensemble des polytopes considérés.

Définition 4.3 *Les graphes $\Gamma(d)$, $\Lambda(d)$ et $\Lambda(d,k)$*

On définit les graphes non-orientés suivants :

- Soit $\Gamma(d)$ le graphe dont les sommets sont les polytopes d-dimensionnels contenus dans $\mathbb{R}^d$ et dans lequel deux polytopes sont reliés par une arête s'il est possible de passer de l'un à l'autre par une insertion (Définition 4.1) ou une suppression (Définition 4.2) d'un sommet.
- Soit $\Lambda(d)$ le sous-graphe induit de $\Gamma(d)$ où l'ensemble des sommets considérés est celui des polytopes entiers.
- Soit $\Lambda(d,k)$ le sous-graphe induit de $\Lambda(d)$ où l'ensemble des sommets est celui des polytopes contenus dans l'hypercube $[0,k]^d$.

Remarquons que $\Lambda(d)$ est un graphe qui est loin d'être régulier : il contient des sommets qui peuvent être de degré fini ou dénombrable. De plus, on verra qu'il existe certains polytopes entiers, comme le cube $[0,1]^d$, dans lesquels il est impossible d'ajouter un sommet et d'autres polytopes, comme les simplexes, dans lesquels aucune suppression n'est possible. Le graphe $\Lambda(d,k)$ permet l'étude des polytopes entiers dans $[0,k]^d$, lesquels sont largement étudiés [AŽ95 ; BL98 ; DM16 ; DMO18 ; DP18 ; KO92].

Afin d'étudier la connexité de ces graphes, il est donc important de comprendre au préalable quand il est possible d'ajouter ou de supprimer un sommet.

1 Opérations interdites

La FIG. 4.1 donne un exemple de sommets que l'on peut ajouter et de sommets qu'il est impossible d'ajouter ou de supprimer à des polygones dans $\mathbb{Z}^2$.

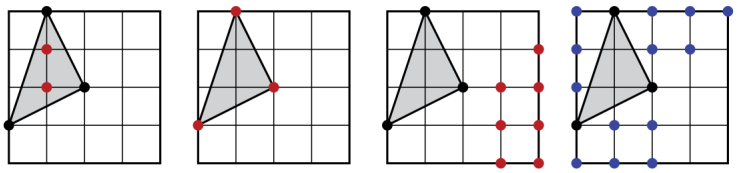


FIG. 4.1 : Soit P le polytope représenté et $\mathcal{V}$ l'ensemble de ses sommets : (1) les sommets en rouge ne peuvent être ajoutés car ils sont à l'intérieur du polytope (le sommet appartient à $P \setminus \mathcal{V}$, (2) on ne peut supprimer un sommet x d'un simplexe car le polytope ne serait plus en dimension pleine (l'enveloppe convexe de $\mathcal{V} \setminus \{x\}$ est en dimension $d-1$), (3) on ne peut ajouter les sommets x en rouge car cela placerait le sommet au centre de l'espace à l'intérieur du polytope ($\mathcal{V} \cup \{x\}$ n'est pas l'ensemble des sommets de son enveloppe convexe), (4) les sommets x en bleu peuvent être ajoutés (l'enveloppe convexe du polytope obtenu a pour sommets $\mathcal{V} \cup \{x\}$)

On va à présent décrire formellement comment détecter si un point de l'espace est dans le cas (3) de la FIG. 4.1. Soit un polytope P dans $\mathbb{R}^d$ et F une face de ce polytope. On note $\text{aff}(F)$ l'enveloppe affine de F. Si F est une facette, alors $\text{aff}(F)$ est un hyperplan dans $\mathbb{R}^d$. On note alors $H_F^-(P)$ le demi-espace de $\mathbb{R}^d$ délimité par $\text{aff}(F)$ et ne contenant pas P. On a $P \cap H_F^-(P) = F$.

On considère à présent un sommet $v \in \mathcal{V}$. L'ensemble des points qui, si on les ajoutait à notre polytope, ferait "disparaître" v de l'enveloppe convexe, forme un cône, que l'on définit ci-dessous. La FIG. 4.2 illustre la définition.

Définition 4.4 *Cône au sommet v*

Pour tout sommet $v \in \mathcal{V}$ d'un polytope P, on définit le cône au sommet v, noté $C_v(P)$, comme

$$C_v(P) = \bigcap_{F \in \mathcal{F}_v} H_F^-(P)$$

où $\mathcal{F}_v$ est l'ensemble des facettes de P qui contiennent v.

Les lemmes suivants indiquent quand il est possible d'insérer ou de supprimer un sommet v.

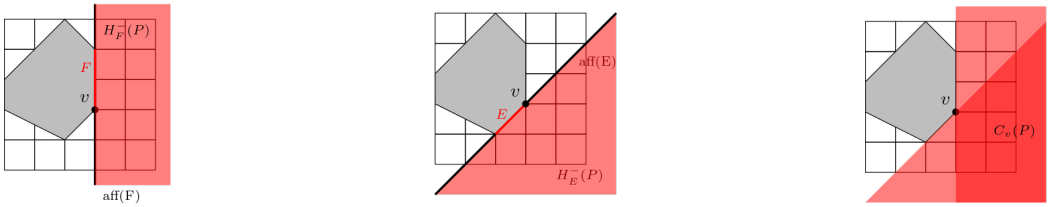


FIG. 4.2 : A gauche : une face F, son enveloppe affine aff(F) et le demi espace $H_F^-(P)$. Au milieu, l'équivalent pour la face E. A droite, le cône $C_v(P)$, intersection des deux demi espaces $H_F^-(P)$ et $H_E^-(P)$

Lemme 4.5

Soit un polytope P dans $\mathbb{R}^d$. Un point $x \in \mathbb{R}^d$ peut être inséré dans P si et seulement si il n'appartient pas à P et que, pour tout sommet v de P, il n'appartient pas à $C_v(P)$.

Une d-pyramide est l'enveloppe convexe de l'union d'un polytope de dimension $(d-1)$, appelé *base*, et d'un sommet v, appelé apex, tel que $v \notin$ aff(P).

Lemme 4.6

Soit un polytope P dans $\mathbb{R}^d$ et $\mathcal{V}$ son ensemble de sommets. Un point $v \in \mathcal{V}$ peut être supprimé de P si et seulement si P n'est pas une d-pyramide d'apex v. Il est donc toujours possible de supprimer un sommet dans un polytope ayant plus de $d+1$ sommets.

2 Connectivité et diamètre de graphes de polytopes

L'idée des théorèmes qui suivent est de montrer la connexité sur des graphes de plus en plus restreints. On étudie donc d'abord celle de $\Gamma(d)$, l'ensemble des polytopes de dimension d, puis $\Lambda(d)$ l'ensemble des polytopes entiers et enfin celle de $\Lambda(d,k)$, les polytopes entiers contenus dans l'hypercube $[0,k]^d$. A chaque étape, on vérifiera si en se restreignant pour tout n à des polytopes à n et $n+1$ sommets, la connexité est déjà vérifiée. Celle-ci implique naturellement la connexité du graphe dans le cas général.

2.1 La connexité du graphe $\Gamma(d)$

Théorème 4.7

Pour tout $n \geq d+1$, les polytopes avec n ou $n+1$ sommets induisent un sous-graphe connexe de $\Gamma(d)$, de diamètre compris entre $4n-2d$ et $6n-2$. Deux polytopes à n sommets sont de distance au plus $6n-4$ dans ce sous-graphe.

L'idée de ce théorème est la suivante : soient deux polytopes P et Q dans $\mathbb{R}^d$. On cherche une façon de transformer P en Q ou inversement, en utilisant les opérations d'ajout et de suppression de sommets, sachant que ces opérations doivent ici s'alterner, afin de rester dans l'ensemble des polytopes ayant n et $n+1$ sommets. On considère sans perte de généralité que P est le polytope contenant le point x ayant la coordonnée x_1 la plus petite possible parmi P et Q. On note

$$\gamma = min\{x_1 \mid x \in P\} \text{ et } \delta = max\{x_1 \mid x \in Q\}$$

Les deux scalaires obtenus permettent de définir deux hyperplans :

$$M = \{x \in \mathbb{R}^d \mid x_1 = \gamma\} \text{ et } N = \{x \in \mathbb{R}^d \mid x_1 = \delta\}$$

La FIG. 4.3 donne un exemple de deux polytopes P et Q, des hyperplans M et N correspondants et des demi-espaces $H_M^-(P)$ et $H_N^-(Q)$. La FIG. 4.4 montre comment passer de l'un à l'autre en appliquant la méthode qui suit. On commence

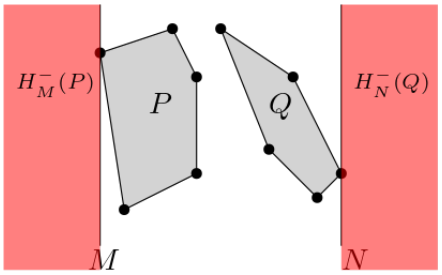


FIG. 4.3 : En haut, un polytope P de départ et un polytope Q d'arrivée. Notons que P et Q peuvent s'intersecter, sans que cela change la méthode.

par transformer P en un polytope P' ayant une facette qui intersecte M et le reste de ses sommets dans $H_M^-(P)$, le demi espace délimité par $\text{aff}(M)$ qui ne contient ni P ni Q. Il faut donc placer $d-1$ points sur l'hyperplan M (en $2d-2$ opérations), afin d'obtenir une facette. Puis, on déplace les $n-d$ points restants de l'autre côté de M (en $2n-2d$ opérations). Notons qu'il est toujours possible d'ajouter un point dans $\mathbb{R}^d$. Il suffit en l'occurrence de le prendre suffisamment proche de la facette que nous venons de créer, comme on peut le voir dans la FIG. 4.4. On fait de même en transformant Q en un polytope Q', ayant une facette qui intersecte N et le reste de ses sommets dans $H_N^-(Q)$, qui ne contient ni Q ni P'. Il est possible de passer de P' à Q' en alternant une suppression d'un sommet de P' et l'ajout d'un sommet de Q'.

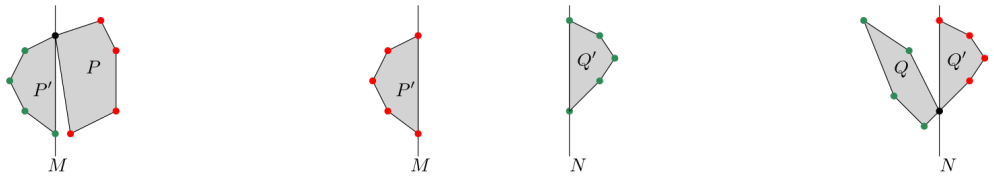


FIG. 4.4 : Sur les 3 schémas, les sommets notés en rouge sont ceux que l'on supprime et ceux en vert ce que l'on ajoute. Il faut nécessairement alterner les ajouts et les suppressions, suivant que le polytope contienne n ou $n+1$ sommets. A gauche : on transforme le polytope P en un polytope P'. Au milieu, on transforme le polytope P' en Q'. A droite, Q' est transformé en Q.

Corollaire 4.8

Le graphe $\Gamma(d)$ est connexe. La distance entre deux polytopes possédant respectivement n et m sommets est au plus $n+m+4d$.

L'idée du corollaire est la suivante : il est possible de transformer chacun des deux polytopes en simplexes en supprimant respectivement $n-d-1$ et $m-d-1$ sommets (soit $n+m-2d-2$ opérations). Passer d'un simplexe a un autre coûte au plus $6(d+1)-4$ opérations, soit $6d+2$.

2.2 La connexité du graphe $\Lambda(d)$

Un changement se produit lorsque l'on passe de $\Gamma(d)$ à $\Lambda(d)$. Il existe désormais des polytopes pour lesquels il est impossible d'ajouter des sommets.

Pour tout $d \geq 2$, si le polytope P que l'on considère est un hypercube de côté 1, alors il est impossible d'ajouter un sommet à ce polytope car tous les points de $\mathbb{Z}^d$ sont inclus dans un des cônes $C_v(P)$. Ce n'est pas le seul polytope dans ce cas.

Lemme 4.9

Pour tout n distinct de 3 et 5, il existe un polygone entier P_n à n sommets tel que aucun point de $\mathbb{Z}^2$ ne peut être inséré dans P_n.

On décrit brièvement à quoi ressemble ce polygone. Soit la fonction $f : \mathbb{Z} \to \mathbb{Z}$ définie par $f(x) = \frac{x(x-1)}{2}$. On considère la ligne polygonale convexe formée par les points $x \in \mathbb{Z}^2$ tels que $x_2 = f(x_1)$. Nous allons à présent construire

un polygone à l'aide de cette ligne. Supposons pour l'instant que n est pair. L'ensemble des sommets du polygone P_n est constitué de :

- un point a tel que $a_1 = \frac{n}{2} - 1$ and $a_2 = f(a_1) + 1$.
- la ligne polygonale convexe formée par les points x tels que $0 \leq x_1 < \frac{n}{2}$ et $x_2 = f(x_1)$.
- les symétriques de la ligne polygonales par rapport au point $\frac{a}{2}$.

Le polygone obtenu est centralement symétrique. Dans le cas où n est impair, le polygone P_n est obtenu en calculant l'enveloppe convexe de P_{n+1} et d'un point c tel que $c_1 = a_1$ et $c_2 = a_2 + 1$. La FIG. 4.5 montre les premiers polygones P_n. Notons qu'il n'en existe pas pour n égal à 5.

Théorème 4.10

Les pentagones et les hexagones induisent un sous-graphe connexe de $\Lambda(2)$.

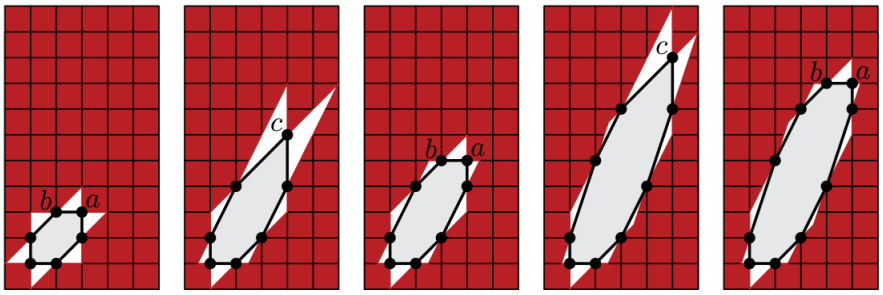


FIG. 4.5 : Les polygones P_n avec n entre 6 et 10 . Les sommets des polytopes en bas à gauche de figure sont de coordonnées $(0, 0)$.

La conséquence du Lemme 4.9 est le Théorème 4.11.

Théorème 4.11

Les sous-graphes induits de $\Lambda(2)$ par les polygones à n et $n + 1$ sommets sont déconnectés lorsque n est différent de 3 et 5.

Ce théorème nous indique que, pour passer d'un polygone à un autre, il va falloir prendre un chemin détourné en s'autorisant à passer par des polytopes plus grands ou plus petits. Fort heureusement, il est toujours possible d'ajouter un sommet à un simplexe, quelle que soit la dimension considérée. On obtient donc le résultat suivant.

Théorème 4.12

Pour tout graphe $d \geq 2$, le graphe $\Lambda(d)$ et son sous-graphe restreint aux polytopes à $d + 1$ et $d + 2$ sommets sont connexes.

Le sous-graphe des simplexes et des polytopes à $d + 2$ sommets étant connexe, il est toujours possible de passer d'un polytope P à un polytope Q en passant par des simplexes intermédiaire et le Corollaire 4.8 sur le diamètre de $\Gamma(d)$ est encore valide sur $\Lambda(d)$.

2.3 La connexité du graphe $\Lambda(d, k)$

Les graphes $\Lambda(d, k)$ sont ceux que l'on cherchait à étudier à l'origine. Puisque les coordonnées des sommets doivent nécessairement être incluses dans $[0, k]^d$, l'ensemble de ces polytopes forme une classe combinatoire et il est donc possible de définir un générateur aléatoire.

Nous avons obtenu les résultats suivants.

Théorème 4.13

Les sous-graphes induits de $\Lambda(d, k)$, restreints aux simplexes et aux polytopes à $d + 2$ sommets, sont connexes.

Afin de montrer la connexité du graphe, il convient de choisir un coin de l'hypercube et de passer de n'importe quel simplexe vers le simplexe le plus petit possible, situé dans ce coin. La Fig. 4.6 illustre cette idée en dimension 2. On commence par transformer le polygone en un simplexe ou un quadrilatère qui possède une arête e qui est soit horizontale, soit verticale. De là, il est possible de construire un quadrilatère dont les arêtes sont soit horizontales, soit verticales. En supprimant le sommet v du simplexe tel que le cône $C_v(P)$ contient le coin de l'hypercube qui nous intéresse, on obtient un triangle, contenant une unique arête oblique, faisant face au coin qui nous intéresse. De là, il est possible de faire une succession d'insertions et de suppressions pour obtenir le simplexe situé dans le coin.

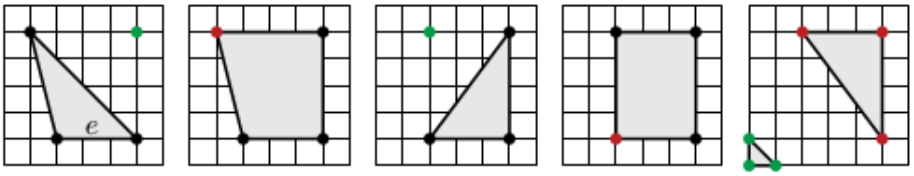


Fig. 4.6 : Passage d'un triangle au plus petit triangle situé dans un coin de l'hypercube $[0,k]^2$.

Théorème 4.14

Les graphes $\Lambda(d,k)$ sont connexes.

L'idée du dernier résultat est qu'il est toujours possible de se ramener à construire un chemin entre un polytope de $\Lambda(d,k)$ et le plus petit simplexe possible situé dans un coin de l'hypercube.

3 Chaînes de Markov sur $\Lambda(d,k)$

On décrit à présent la matrice de transitions associée à $\Lambda(d,k)$. Pour tout polytope entier $P \in \Lambda(d,k)$, on suppose que l'on a tiré aléatoirement et uniformément un point $x \in [0,\dots,k]^d$. On distingue alors les 3 cas suivants :

- x est un des sommets de P et peut être supprimé en respectant la Définition 4.2. Soit Q le polytope obtenu. La probabilité de passer de P à Q est $\frac{1}{(k+1)^d}$.
- $x \notin P$ et x peut être ajouté en respectant la Définition 4.1. Soit Q le polytope obtenu. La probabilité de passer de P à Q est $\frac{1}{(k+1)^d}$.
- dans tous les autres cas, le polytope n'est pas affecté.

Algorithme 7 : Générateur aléatoire de polytope dans $\Lambda(d,k)$

Entrées : Une dimension d et k le côté de l'hypercube.
Sorties : Un polytope aléatoire de $\Lambda(d,k)$
$P \leftarrow$ Engendrer aléatoirement et uniformément un simplexe dans $[0,k]^d$;
tant que *On est pas assez proche de la distribution stationnaire* **faire**
  $x \leftarrow$ Engendrer aléatoirement et uniformément un point de $[0,k]^d$;
  si $x \in \mathcal{V}$ **alors**
    $Q \leftarrow conv(\mathcal{V} \setminus \{x\})$;
    si Q *est en dimension* d **alors**
      $P \leftarrow Q$;
  sinon
    $Q \leftarrow conv(\mathcal{V} \cup \{x\})$;
    si *l'ensemble des sommets de* Q *est* $\mathcal{V} \cup \{x\}$ **alors**
      $P \leftarrow Q$;
retourner P ;

Théorème 4.15

Pour tout $d \geq 2$, et tout $k > 0$, la chaîne de Markov correspondant à l'algorithme 7 est irréductible, apériodique et réversible.

L'irréductibilité de la chaîne est donnée par le théorème 4.14. Il est facile de voir que la chaîne est apériodique : la chaîne étant irréductible, pour montrer que le pgcd des longueurs des cycles est 1, il suffit de trouver un unique cycle de taille 1. Or pour tout simplexe, il est impossible de supprimer l'un de ses sommets. Il y aura donc une boucle sur ces polytopes. La réversibilité est également assez évidente au vue de la définition : la probabilité de passer d'un polytope P à un polytope Q est la même que de passer de Q à P et est égale à $\frac{1}{(k+1)^d}$.

3.1 Temps de mélange

Comme nous l'avons mentionné dans l'interlude, il est possible d'obtenir des bornes inférieure sur le temps de mélange à partir du diamètre du graphe associé à la chaîne de Markov (Propriété 2.2). Pour passer d'un polytope à un autre, on supprime les sommets jusqu'à obtenir un simplexe, que l'on transforme pour le placer dans un coin de l'hypercube, avant de refaire le chemin inverse vers le polytope d'arrivée. Le diamètre est donc lié au nombre maximal de sommets dans un polytope de $\Lambda(d, k)$. Dans [AŽ95], les auteurs montrent que le nombre maximal de sommets dans un polygone convexe inclus dans $[0, k]^2$ est

$$12\left(\frac{k}{2\pi}\right)^{2/3} + \mathcal{O}(k^{1/3}\log(k))$$

Théorème 4.16

En dimension 2, pour tout $\varepsilon > 0$, il existe une constante c tel que le temps de mélange $t_{mix}(\varepsilon)$ est supérieur à $ck^{\frac{2}{3}}$

Dans le cas général, Barany et Larman [BL98] on montré que le nombre de sommets dans l'enveloppe convexe des points à coordonnées entières contenus dans une sphère de dimension $d \geq 2$ et de rayon r est $\Theta(r^{d\frac{d-1}{d+1}})$. En considérant non pas une sphère de rayon r mais un hypercube de côté $k = r$, on obtient la borne inférieure suivante.

Théorème 4.17

En dimension $d \geq 2$, pour tout $\varepsilon > 0$, il existe une constante c tel que le temps de mélange $t_{mix}(\varepsilon)$ est supérieur à $ck^{d\frac{d-1}{d+1}}$.

3.2 Expérimentations et conjectures

Nous avons utilisé le générateur aléatoire à base de chaîne de Markov et effectué plusieurs observations des statistiques des polytopes engendrés, avec lesquelles nous avons obtenus les conjectures suivantes.

Conjecture 4.1

Le nombre moyen de sommets d'un polygone entier contenu dans $[0, k]^2$ est

$$\mathbb{E}[n] \geq 6\left(\frac{k}{2\pi}\right)^{2/3}$$

Notons que cette borne correspond au diamètre (moitié du nombre maximal de sommets) des zonotopes primitifs [DMO18]. Les auteurs conjecturent que ces zonotopes maximisent le diamètre des polytopes. En d'autres termes, un polygone aléatoire possède un nombre moyen de sommets égal à la moitié du nombre maximal.

Conjecture 4.2

L'aire moyenne d'un polygone entier contenu dans $[0, k]^2$ est

$$\mathbb{E}[a] \geq \frac{3}{4}k^2$$

Cette deuxième conjecture repose principalement sur une observation empirique et tend à dire que les polygones aléatoires remplissent une bonne partie du carré dans lesquels ils sont inclus.

4 Conclusion

Comme mentionné dans l'introduction, ces travaux sont issus d'une collaboration avec Lionel Pournin dans le cadre du co-encadrement de la thèse de Rado Rakotonarivo. Je ne souhaite en revanche pas poursuivre de travaux dans cette direction, la géométrie en dimensions supérieures ne faisant pas partie de mes domaines de prédilection.

Interlude : une méthodologie pour l'analyse en moyenne

Comme cela a pu être précisé dans l'introduction, les résultats théoriques sur la complexité moyenne des algorithmes peuvent permettre d'expliquer le comportement de ces derniers en pratique, sans que la distribution considérée semble elle même réaliste.

La thèse soutenue dans ce manuscrit est la suivante : si la complexité moyenne prévoit un comportement de l'algorithme similaire à celui observé en pratique, on peut supposer qu'il existe une propriété commune aux deux situations, déterminante pour le comportement de l'algorithme.

Afin de pouvoir mettre en avant ces propriétés, je propose d'utiliser la méthodologie suivante : l'idée est de considérer la complexité moyenne sur un ensemble de distributions de probabilités au lieu de se concentrer sur une seule.

Soit $\mathcal{E}_n$ l'ensemble des entrées de taille n de l'algorithme. Soit $\mathcal{D}_n$ l'ensemble des distributions possibles sur $\mathcal{E}_n$ et $\mathcal{D} = \bigcup_{n\geq 1} \mathcal{D}_n$. Soit $CoutMoyen : \mathcal{D} \to \mathbb{R}$ une fonction qui retourne le coût moyen de l'algorithme selon une distribution $\pi \in \mathcal{D}$. Soit $D_n \subseteq \mathcal{D}_n$ un ensemble de distributions partageant une même propriété et μ une mesure sur D_n. On veut étudier l'intégrale suivante :

$$\int_{\pi \in D_n} CoutMoyen(\pi) d\mu(\pi)$$

Autrement dit, on s'intéresse à une moyenne de moyenne. Il convient maintenant de déterminer quel ensemble D_n pourrait être intéressant à étudier.

Afin de pouvoir parler de complexité, il convient de définir proprement une notion d'asymptotique. Soit $(D_n)_{n\in\mathbb{N}}$ une famille telle que chaque D_n est un ensemble de distributions sur $\mathcal{E}_i$ et $(\mu_n)_{n\in\mathbb{N}}$ une famille de mesures sur D_n. La complexité moyenne que nous considérons est donnée par le comportement asymptotique de l'intégrale suivante :

$$\int_{\pi \in D_n} CoutMoyen(\pi) d\mu_n(\pi_n)$$

Dans le chapitre 5, l'un des modèles probabilistes considéré reprend cette idée d'une intégrale, où des distributions produits construites à partir de lois de Bernoulli sont considérées. On verra dans ce chapitre l'intérêt d'utiliser une intégrale et une mesure : l'idée est de n'avoir à fixer que la mesure μ pour déterminer directement l'ensemble D_n, ce qui constituera une simplification par rapport à un autre modèle proposé.

Dans le chapitre 6, on considère que D_n est fini. La mesure est alors remplaçable par une distribution de probabilité et l'intégrale peut être remplacée par une simple somme.

$$\sum_{\pi \in D_n} \mathbb{P}(\pi) CoutMoyen(\pi)$$

De plus, dans ce chapitre, on considérera que $\mathbb{P}(\pi) = \frac{1}{|D_n|}$, c'est à dire la distribution uniforme sur D_n.

Comme nous le verrons dans les chapitres suivants, le calcul de l'intégrale peut être largement simplifié dans certains contextes. Supposons que toutes les distributions dans un ensemble D_n partagent une même propriété telle que $\forall \pi, \pi' \in D_n$, on a $CoutMoyen(\pi) = CoutMoyen(\pi')$. On dira dans ce cas que la propriété est *déterminante*. Le calcul est de nouveau simplifié car il est possible de remplacer l'intégrale par l'évaluation de $CoutMoyen$ en n'importe quelle distribution de D_n.

Supposons maintenant que la propriété considérée sur les distributions est l'image d'une application. Soit $T : \mathcal{D} \to [0, +\infty[$ une application qui prend en entrée une distribution. Pour une valeur fixée t, appelée *cible*, on définit l'ensemble $D_{t,n} \in \mathcal{D}$ l'ensemble des distributions sur $\mathcal{E}_n$ telle que $\forall \pi \in D_{t,n}, T(\pi) = t$.

Soit $CoutParametre(t, n) = \int_{\pi \in D_{t,n}} \mathbb{P}(\pi) CoutMoyen(\pi) d\pi$ la fonction qui prend en entrée une cible et renvoie le coût moyen de l'algorithme sur $D_{t,n}$.

Définition 4.18 *Fonction déterminante*

Une fonction $T : \mathcal{D} \to [0, +\infty[$ est *déterminante* si $\forall t, \forall n, \forall \pi, \pi' \in D_{t,n}$, on a

$$CoutMoyen(\pi) = CoutMoyen(\pi')$$

On rappelle que le coût maximal (resp. minimal) d'un algorithme sur un ensemble d'entrées de taille n est noté p_n (resp. m_n).

Définition 4.19 *Fonction intéressante*

Une fonction $T : \mathcal{D} \to [0, +\infty[$ est dite *intéressante* si :
- $CoutParametre(t, n)$ est continue et monotone pour tout n fixé.
- il existe une valeur t telle que $CoutParametre(t, n) = \theta(p_n)$ ou $CoutParametre(t, n) = \theta(m_n)$
- il existe une valeur t telle que $CoutParametre(t, n) = \Theta(CoutMoyen(\pi))$, où π est la distribution uniforme sur $\mathcal{E}_n$.

Dans le chapitre 6, on considère deux fonctions sur $\mathcal{D}$: l'entropie de Shannon h et l'entropie de collision κ.

$$H(\pi) = -\sum_{i=1}^{k} \pi_i \log_2(\pi) \text{ and } \kappa(\pi) = -\log_2\left(\sum_{i=1}^{k} \pi_i^2\right)$$

Comme on le verra, les deux fonctions sont *intéressantes* pour l'analyse du coût moyen de l'algorithme naïf de recherche de séquence, mais seule l'entropie de collision est *déterminante*.

5

Analyse en moyenne sur les hypergraphes

Article(s) associé(s).
- *Julien David, Loick Lhote, Arnaud Mary and Francois Rioult, An Average Study of Hypergraphs and their minimal transversals.. Theor. Comput. Sci. 596 : 124-141 (2015).*
- *Julien David, Mostafa Gholami, Loick Lhote, On the average complexity of Berge Algorithm, soumis à SODA 2025.*

Mostafa Gholami est actuellement en doctorat, sous la direction de Loick Lhote et co-encadré par moi.

Les hypergraphes sont une généralisation de la notion de graphe, explicitement nommée par Claude Berge [Ber89] et définie comme suit.

Définition 5.1 *Hypergraphe*

Un hypergraphe est un couple $\mathcal{H} = (V, E)$ tel que
- $V = \{1, \dots, n\}$ est un ensemble de sommets
- $E = \{H_1, \dots, H_m\}$ est l'ensemble de ses hyperarêtes, avec $H_i \subseteq V$, pour tout $i \in \{1, \dots, m\}$.

Un hypergraphe est dit **simple** si aucune hyperarête n'est inclue dans une autre.
Berge définit également la notion de traverse, de traverse minimale, et les problèmes suivants.

Définition 5.2 *Traverse et traverse minimale*

Soit un hypergraphe $\mathcal{H} = (V, E)$. Une *traverse* est un ensemble de sommet $T \subseteq V$ qui intersecte toutes les hyperarêtes de $\mathcal{H}$, c'est à dire :

$$\forall H \in E, H \cap T \neq \emptyset$$

Une traverse T est *minimale* s'il n'existe pas de traverse T' telle que $T' \subsetneq T$.

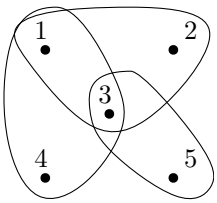


FIG. 5.1 : Dans l'exemple ci-dessus, l'hypergraphe représenté est $\mathcal{H} = (V, E)$ tel que $V = \{1, 2, 3, 4, 5\}$ et $E = \{\{1, 2, 3\}, \{1, 3, 4\}, \{3, 5\}\}$. L'ensemble $\{2, 3\}$ est une traverse, mais elle n'est pas minimale, car $\{3\}$ est également une traverse. L'ensemble $\{1, 5\}$ est également une traverse minimale.

Définition 5.3 *Hypergraphe Transversal*

Soit un hypergraphe simple $\mathcal{H} = (V, E)$. L'ensemble des traverses minimales de $\mathcal{H}$, noté $TrMin(\mathcal{H})$ forme un nouvel hypergraphe, appelé *hypergraphe transversal*, dont l'ensemble des sommets est également V. On a :

$$\mathcal{H} = TrMin(TrMin(\mathcal{H}))$$

L'hypergraphe transversal est également appelé *hypergraphe dual.*

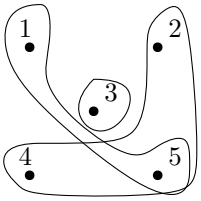


FIG. 5.2 : L'exemple ci-dessus est l'hypergraphe transversal de celui de la Figure 5.1

Problème de décision 5.4 *Transversal Hypergraph Decision(THD)*

Soient deux hypergraphes $\mathcal{H}_1$ et $\mathcal{H}_2$. $\mathcal{H}_2$ est-il l'hypergraphe transversal de $\mathcal{H}_1$?

Problème d'énumération 5.5 *Transversal Hypergraph Generation (THG)*

Soit un hypergraphe $\mathcal{H}$, calculer son hypergraphe transversal.

L'hypergraphe dual peut être exponentiellement plus grand que l'hypergraphe passé en entrée. Il est donc d'usage pour THG , comme dans la plupart des problèmes d'énumération, d'évaluer la complexité non pas en fonction de la taille de l'entrée, mais en fonction de la taille de l'entrée et de la sortie, c'est à dire de $\mathcal{H}_1$ et $\mathcal{H}_2$ si on se rapporte à THD .

Ces deux problèmes ont des applications directes en logique des propositions. En effet, le problème THD est équivalent à se demander si deux formules, l'une en forme normale conjonctive (CNF) et l'autre en forme normale disjonctive (DNF), sont équivalentes. Le problème est dans co-NP, car il suffit d'exhiber une traverse minimale de l'un des hypergraphes et de vérifier qu'elle n'appartient pas en tant qu'hyperarête à l'autre hypergraphe, les deux vérifications pouvant se faire en temps polynomial. Dans [FK96], les auteurs montrent que le problème peut être résolu en temps $N^{o(\log(N))}$, où N est la somme des tailles de $\mathcal{H}_1$ et $\mathcal{H}_2$. L'enjeu est donc de taille car les trois cas de figures sont possibles : soit il existe un algorithme polynomial pour résoudre ce problème, soit il existe une borne inférieure pour ce problème en temps quasi-polynomial ce qui implique que soit ce problème est co-NP complet et tous les problèmes de cette classe de complexité peuvent être résolus en temps quasi-polynomial, soit le problème appartient à une classe entre P et co-NP, ce qui implique que P est différent de co-NP et de NP. Plus largement, ces problèmes ont attiré une attention considérable ces 30 dernières années, de par leur large spectre d'applications dans de nombreux domaines, incluant l'intelligence artificielle et la logique [EG02], biologie computationnelle [Dam06 ; KSG07 ; HKS08], la cryptographie [KP04], la théorie des bases de données [MR92], la fouille de données [Gun+03 ; AIS93 ; BG03], le calcul distribué [GB85], le e-commerce [BZ06], l'apprentissage automatique [Gun+97], l'optimisation mathématique [Kha01 ; Kha+08], les systèmes de communication mobiles [SS98], le Web sémantique [KLM07], la topologie [Dum+03], la combinatoire [Rea78] et la théorie des jeux [BG03]. Enfin, le problème d'énumération des dominants minimaux dans les graphes est équivalent au problème THG [BLS99 ; Kan+14].

Le chapitre contient des résultats sur deux algorithmes permettant de résoudre le problème THG : l'algorithme de Berge et MT-Miner. Dans la section 1, on introduit trois modèles d'hypergraphes aléatoires, de plus en plus généraux : $\mathcal{HG}(m,n,p)$, $\mathcal{HG}(m,n,\mathbf{v})$ et $\mathcal{HG}(m,n,\mu)$. Le modèle le plus général, $\mathcal{HG}(m,n,\mu)$, correspond à la méthodologie décrite dans l'interlude. Si nous n'avons pas encore obtenu d'asymptotiques dans ce modèle, nous avons toutefois une série génératrice à partir de laquelle nous espérons pouvoir prochainement obtenir des résultats. La section 1.1 montre, à partir de résultats expérimentaux, la pertinence de nos modèles les plus généraux. La section 2 présente l'algorithme de Berge et MT-Miner, ainsi que les paramètres d'intérêts pour l'analyse de leur comportement. La section 3 est basée sur notre premier article et une approche probabiliste. On obtient des bornes supérieures sur le nombre moyen de traverses, d'irredondants (le paramètre d'intérêt de MT-Miner) et de traverses minimales dans le modèle $\mathcal{HG}(m,n,p)$, ainsi que des résultats moins précis dans le modèle $\mathcal{HG}(m,n,\mathbf{v})$. On en déduit des résultats sur la complexité moyenne de l'algorithme MT-Miner. La section 4 est basée sur le second article et un ensemble de techniques de combinatoire analytique. On obtient ensuite des résultats plus précis que dans la section 3, dans le modèle $\mathcal{HG}(m,n,p)$ et dans certaines versions du modèle $\mathcal{HG}(m,n,\mathbf{v})$.

1 Modèles probabilistes sur les hypergraphes

Dans cette section, on définit trois modèles d'hypergraphes aléatoires. Un hypergraphe H peut être vu comme une matrice binaire $M(\mathsf{H}) = (\chi_{i,j}(\mathsf{H})Not)_{i=1..m,j=1..n}$, appelée matrice d'incidence de H, avec $\chi_{i,j}(\mathsf{H}) = 1$ si et seulement si le sommet v_j appartient à l'hyperarête e_i. Notons que, dans les cas que nous allons étudier, la probabilité que l'hypergraphe ne contiennent pas d'arêtes multiples (que deux lignes de la matrice soient égale), tend vers 1 dans les différents modèles.

Dans le premier modèle aléatoire, les colonnes et les lignes de la matrice sont indépendantes. Il peut être vu comme l'adaptation du modèle d'Erdös-Renyi [ER59] sur les hypergraphes.

Définition 5.6 *$\mathcal{HG}(m,n,p)$ random model*

Soit $p \in [0,1]$ un paramètre. Dans le modèle $\mathcal{HG}(m,n,p)$, la famille de variables aléatoires $(\chi_{i,j}(\mathsf{H}))_{i=1..m,j=1..n}$ forme une famille de variables aléatoires indépendantes et identiquement distribuées telle que pour tout (i,j), $\chi_{i,j}$ suit une loi de Bernoulli de paramètre p.

Le second modèle généralise le précédent. Toutes les lignes de la matrice, correspondant aux hyperarêtes, suivent la même distribution. Pour les colonnes, le degré de chaque sommet v_j suit une loi binomiale de paramètre (m,π_j). Le degré moyen de v_j est donc contrôlé et égal à $m\pi_j$.

Définition 5.7 *$\mathcal{HG}(m,n,\mathbf{v})$ random model*

Soit une séquence $\mathbf{v} = (\mathbf{v}_1,\ldots,\mathbf{v}_n)$ in $[0,1]^n$ de paramètres. Dans le modèle $\mathcal{HG}(m,n,\mathbf{v})$, la famille de variables aléatoires $(\chi_{i,j}(\mathsf{H}))_{i=1..m,j=1..n}$ forme une famille de variables aléatoires indépendantes telle que pour tout (i,j), $\chi_{i,j}$ suit une loi de Bernoulli de paramètre $\mathbf{v}_j$.

Le troisième modèle que l'on considère est assez puissant pour capturer la complexité de données réelles, mais est dans un sens plus simple, car toutes les colonnes ont le même comportement probabiliste.

Définition 5.8 *$\mathcal{HG}(m,n,\mu)$ random model*

Soit μ une mesure de probabilité sur $[0,1]$. Dans le modèle $\mathcal{HG}(m,n,\mu)$, la matrice d'incidence $M(\mathsf{H})$ est engendrée selon la procédure suivante :

1. Engendrer un vecteur aléatoire $\mathbf{v} = (\mathbf{v}_1,\ldots,\mathbf{v}_n)$ où $(\mathbf{v}_1,\ldots,\mathbf{v}_n)$ forme une famille de variables aléatoires i.i.d. de mesure de probabilité μ ;
2. Engendrer $M(\mathsf{H})$ selon le modèle $\mathcal{HG}(m,n,\mathbf{v})$.

Dans le modèle $\mathcal{HG}(m,n,\mu)$, la distribution des degrés est contrôlée. L'objectif étant de décrire l'influence des degrés sur le nombre de traverses minimales.

Ces modèles sont étroitement liés. Soit $\mathbf{Pr}_{m,n,\mu}[\mathsf{H}]$ la probabilité de tirer un hypergraphe dans $\mathcal{HG}(m,n,\mu)$ et $\mathbf{Pr}_{m,n,\mathbf{v}}[\mathsf{H}]$ la probabilité de tirer un hypergraphe dans $\mathcal{HG}(m,n,\mathbf{v})$

$$\mathbf{Pr}_{m,n,\mu}[\mathsf{H}] = \int_{[0,1]^n} \mathbf{Pr}_{m,n,\mathbf{v}}[\mathsf{H}] d\mu_{\otimes n}(\mathbf{v}) \tag{5.1}$$

avec $\mu_{\otimes n}$ la mesure produit $\mu \otimes \cdots \otimes \mu$ (n fois). De plus, si on considère la mesure de Dirac $\mu = \delta_p$, alors $\mathcal{HG}(m,n,\delta_p)$ est exactement le modèle $\mathcal{HG}(m,n,p)$. Notons que, dans ce chapitre, les résultats présentés ne concernent que les modèles $\mathcal{HG}(m,n,p)$ et $\mathcal{HG}(m,n,\mathbf{v})$. En revanche, les contours de travaux futurs autour de $\mathcal{HG}(m,n,\mu)$ sont dessinés.

1.1 Données réelles versus modèles

Une question essentielle à se poser après avoir défini ces modèles, est leur pertinence. En prenant des exemples issus de la *Frequent Itemset Mining Dataset Repository*[1], on obtient une réponse en demie teinte.

Les histogrammes suivants représentent respectivement le nombre d'hyperarêtes dans lesquelles apparaissent chaque sommet et le nombre d'hyperarêtes d'une taille donnée. Dans la section 3.2, on définit les notions de sommets ubiquitaires, c'est-à-dire présents dans presque toutes les hyperarêtes et les sommets rares, qui n'apparaissent que dans une proportion faible d'hyperarêtes. On verra par la suite que ces sommets jouent un rôle important dans le comportement de l'algorithme MT-Miner.

L'hypergraphe *mushroom.dat* est un hypergraphe dit k-uniforme : toutes les hyperarêtes sont de cardinal $k = 23$. L'existence de ce type de données valide le choix de plusieurs études qui se sont concentrées sur les hypergraphes k-uniformes [AM02], [DF10] et [Lel12]. Les modèles présentés dans cette section ne permettent pas de modéliser les graphes k-uniformes.

L'hypergraphe *pumsbstar.dat* ne contient pas de sommets ubiquitaires mais contient principalement des sommets rares.

L'hypergraphe *accident.dat* contient peu de sommets ubiquitaires et de nombreux sommets rares.

[1] http ://fimi.ua.ac.be/data/

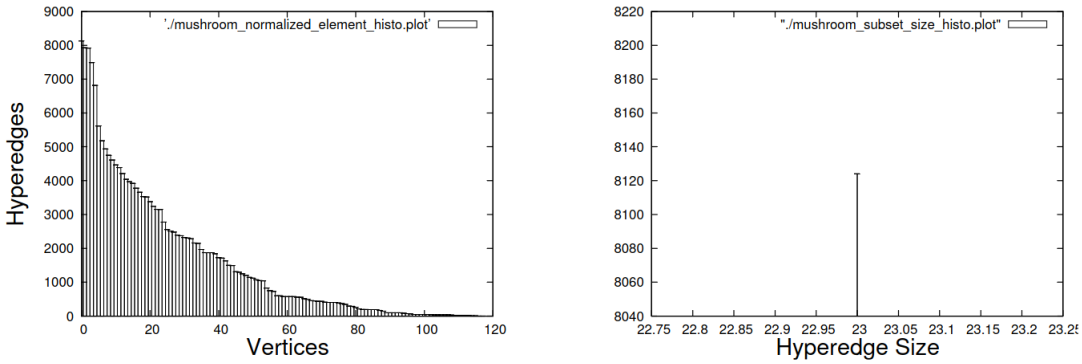


FIG. 5.3 : mushroom.dat (119 sommets, 8124 hyperarêtes)

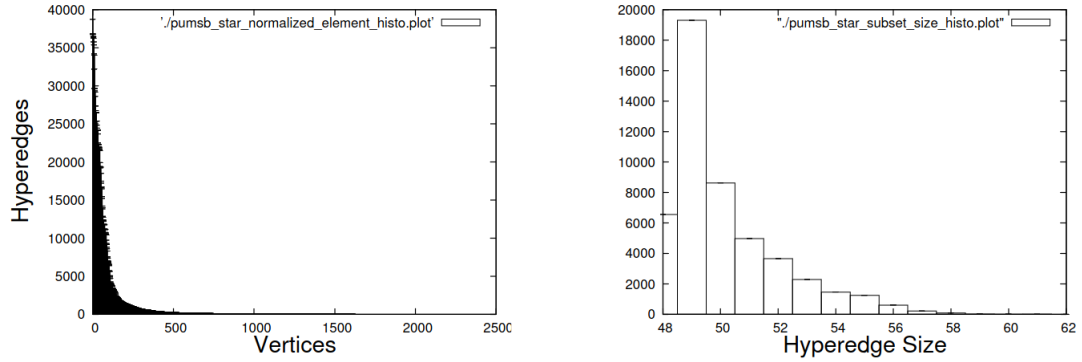


FIG. 5.4 : pumsbstar.dat (7116 sommets, 49046 hyperarêtes)

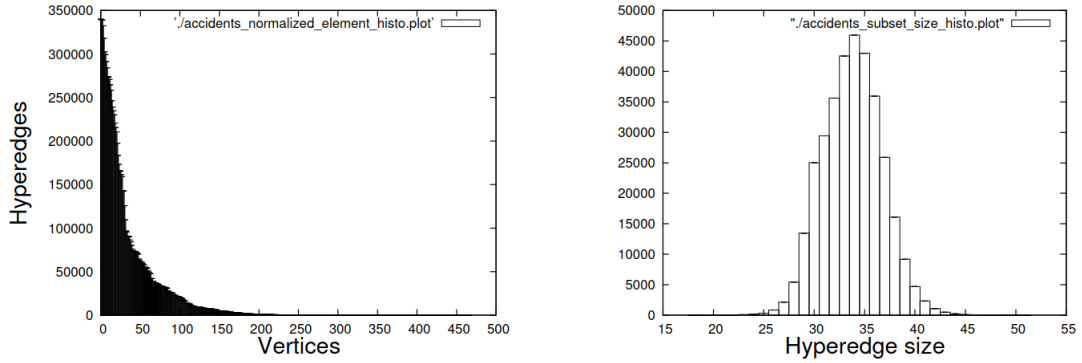


FIG. 5.5 : accidents.dat (468 sommets, 340183 hyperarêtes)

Dans l'hypergraphe *pumpstar.dat*, tous les sommets sont rares. Dans les deux derniers hypergraphes, la taille des hyperarêtes semble approximable par une loi normale.

Les modèles $\mathcal{HG}(m, n, \mathbf{v})$ et $\mathcal{HG}(m, n, \mu)$ semblent donc suffisants pour capturer la complexité des trois derniers hypergraphes, mais le modèle $\mathcal{HG}(m, n, p)$ est trop simpliste. Dans notre approche probabiliste, on verra en revanche que les résultats obtenus sur le modèle $\mathcal{HG}(m, n, p)$ permettent d'en obtenir sur le modèle $\mathcal{HG}(m, n, \mathbf{v})$.

2 Algorithmes d'énumération des traverses minimales

Nous invitons le lecteur à regarder le chapitre 5 de la thèse de Mathias Hagen [Hag08], qui contient la description des différents algorithmes. On se contente ici de présenter l'algorithme de Berge [Ber89], dont on donnera la complexité

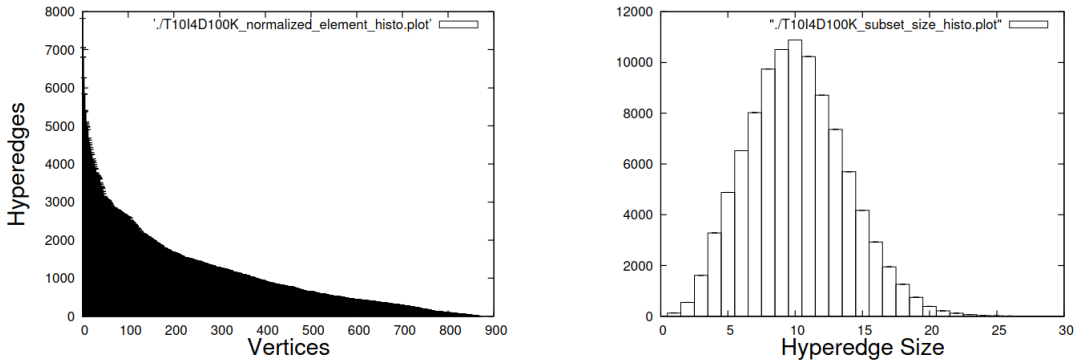


FIG. 5.6 : T10I4D100K.dat (999 sommets, 100000 hyperarêtes)

moyenne dans la section 4 et l'algorithme MT-Miner [HBC07], dont on donnera la complexité moyenne dans la section 3.

2.1 Algorithme de Berge [Ber89]

Soient deux hypergraphes $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ et $\mathcal{H}' = (\mathcal{V}', \mathcal{E}')$ où $\mathcal{E} = \{e_1, \cdots e_m\}$ et $\mathcal{E}' = \{e'_1, \cdots e'_{m'}\}$. Il existe deux notions d'"unions" différentes, à savoir $\mathcal{H} \cup \mathcal{H}'$ et $\mathcal{H} \vee \mathcal{H}'$ définies comme suit.

$$\mathcal{H} \cup \mathcal{H}' = (\mathcal{V} \cup \mathcal{V}', \{e_1, e_2, \cdots, e_m, e'_1, e'_2, \cdots, e'_{m'}\})$$
$$\mathcal{H} \vee \mathcal{H}' = (\mathcal{V} \cup \mathcal{V}', \{e_i \cup e'_j \mid i = 1, 2, \cdots m, \quad j = 1, 2, \cdots m'\})$$

Berge utilise ces deux opérations pour décrire la décomposition suivante.

Proposition 5.1

Soient $\mathcal{H}$ et $\mathcal{H}'$ deux hypergraphes. On a

$$TrMin(\mathcal{H} \cup \mathcal{H}') = \min(Tr(\mathcal{H}) \vee Tr(\mathcal{H}'))$$

où min est l'opération consistant à ne conserver que les hyperarêtes minimales par inclusion.

Autrement dit, il est possible de construire l'hypergraphe transversal d'un hypergraphe $\mathcal{H} \cup \mathcal{H}'$ dès lors qu'on connaît les hypergraphes transversaux de $\mathcal{H}$ et $\mathcal{H}'$. Berge utilise ensuite cette décomposition pour décrire un algorithme incrémental, où l'on ajoute les hyperarêtes de l'hypergraphe initial une par une, créant ainsi m résultats intermédiaires. Pour un hypergraphe $\mathcal{H} = (V, E = (H_1, \ldots, H_m))$, on note $\mathcal{H}_i = (V, E = (H_1, \ldots, H_i))$ l'hypergraphe composé des m premières hyperarêtes.

Algorithme 8 : Algorithme de Berge

Entrées : un hypergraphe $\mathcal{H} = (V, E = (H_1, \ldots, H_m))$
Sorties : l'ensemble des traverses minimales de $\mathcal{H}$
$Tr(\mathcal{H}_1) \leftarrow (H_1, \{\{v\} \mid v \in H_1\})$;
pour tous les $i \in \{2, \ldots, m\}$ **faire**
  $Tr(\mathcal{H}_i) = \min(Tr(\mathcal{H}_{i-1}) \vee (H_i, \{\{v\} \mid v \in H_i\}))$;
retourner $Tr(\mathcal{H}_m)$;

Si cette description non détaillée de l'algorithme de Berge donne l'impression d'une grande simplicité, elle laisse plusieurs questions ouvertes qu'il est impératif de résoudre au moment de l'implantation.

- L'algorithme, tel quel, est non-déterministe. L'ordre dans lequel les hyperarêtes sont considérées n'est pas précisé et, si tout ordre est valide, le comportement de l'algorithme peut changer radicalement.
- la structure de données utilisée pour stocker les résultats intermédiaires Res_i n'étant pas précisée, le coût en temps et en espace pour ne stocker que des ensembles minimaux par inclusion n'est pas clair.

Toutefois, Takata [Tak07] exhibe en 2007 une famille d'hypergraphes pour lesquels, quel que soit l'ordre utilisé, le nombre de solutions intermédiaires obtenues à l'avant dernière opération est toujours super-polynomial dans la taille de l'entrée et de la sortie. La complexité de l'algorithme de Berge est au moins par $N^{\Omega(\log \log N)}$, où N est la taille de l'entrée

plus celle de la sortie. L'année suivante, Boros *et al.* [BEM08] montrent que l'algorithme est polynomial dans les cas suivants :

- le cardinal de toute hyperarête est bornée par une constante,
- ou le degré de tout sommet est borné par une constante,
- ou pour toute intersection de k hyperarêtes, la taille de l'ensemble obtenu est inférieure à r, où k et r sont des constantes. Notons que lorsque k et r sont d'ordre $\log(n)$, on obtient une borne quasi-polynomiale.

Ils donnent également un algorithme permettant d'ordonner les hyperarêtes du graphe de telle sorte que l'algorithme de Berge soit quasi-polynomial si l'hypergraphe est k-conforme (voir définition dans [Ber89], page 30, pour plus de détail).

Intéressons nous à présent à l'efficacité de l'algorithme. Pour chaque traverse minimale $M \in Tr(\mathcal{H}_{i-1})$, l'algorithme ajoute un sommet $v \in H_i$ et vérifie la minimalité, ce qui peut être réalisé en temps $\Theta(|M| \times i)$ (la traverses étant de taille au plus i) et en espace $\Theta(|M|)$, en utilisant une table de hachage. Puisque $|M| \leq min\{n, i\}$, le coût de l'algorithme de Berge sur un hypergraphe $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, notée $B(\mathcal{H})$, est compris dans l'intervalle suivant :

$$\sum_{i=2}^{m} |Tr(\mathcal{H}_{i-1})| \cdot |H_i| \leq B(\mathcal{H}) \leq \sum_{i=2}^{m} |Tr(\mathcal{H}_{i-1})| \cdot |H_i| \cdot min\{n, i\} \cdot i$$

Dans le modèle $\mathcal{HG}(m, n, \mathbf{v})$, on note $\mathbb{E}_{\mathbf{v}}[Tr_{m,n}]$ le nombre moyen de traverses minimales d'un hypergraphe à m hyperarêtes et n sommets et $\mathbb{E}_{\mathbf{v}}[B(\mathcal{H})]$ la complexité moyenne de l'algorithme de Berge. On a :

$$\sum_{i=1}^{m-1} \mathbb{E}_{\mathbf{v}}[Tr_{i,n}] \leq \mathbb{E}_{\mathbf{v}}[B(\mathcal{H})] \leq n \cdot m \cdot min\{n, m\} \sum_{i=1}^{m-1} \mathbb{E}_{\mathbf{v}}[Tr_{i,n}]$$

Soit $c_{m,X}$ la probabilité d'un ensemble X de cardinal ℓ d'être une traverse minimale dans un hypergraphe à m hyperarêtes. On a :

$$\mathbb{E}_{\mathbf{v}}[Tr_{m,n}] = \sum_{\ell=1}^{min\{n,m\}} \sum_{\substack{X \subseteq \{1,\ldots,n\} \\ |X|=\ell}} c_{m,X}$$

Dans le modèle $\mathcal{HG}(m, n, p)$, la formule se simplifie. Soit $\mathbb{E}_p[Tr_{m,n}]$ le nombre moyen de traverses minimales et $c_{m,\ell}$ la probabilité pour un ensemble de cardinal ℓ d'être une traverse minimale. On a :

$$\mathbb{E}_p[Tr_{m,n}] = \sum_{\ell=1}^{min\{n,m\}} \binom{n}{\ell} c_{m,\ell}$$

On peut à présent encadrer le coût moyen de l'algorithme de Berge :

$$\sum_{i=1}^{m-1} \mathbb{E}_p[Tr_{i,n}] = \sum_{i=1}^{m-1} \sum_{\ell=1}^{min\{n,m\}} \binom{n}{\ell} c_{i,\ell} = \sum_{\ell=1}^{min\{n,m\}} \binom{n}{\ell} \sum_{i=1}^{m-1} c_{i,\ell}. \tag{5.2}$$

On nomme *somme cumulative*, notée $\bar{c}_{m,\ell}$, la valeur $\bar{c}_{m,\ell} = \sum_{i=1}^{m-1} c_{i,\ell}$. Afin d'évaluer la complexité de l'algorithme de Berge, on s'intéressera donc à la somme suivante :

$$\sum_{i=1}^{m-1} \mathbb{E}_{\mathbf{v}}[Tr_{i,n}] = \sum_{\ell=1}^{min\{n,m\}} \binom{n}{\ell} \bar{c}_{m,\ell}$$

2.2 Algorithme de Hébert, Bretto et Crémillieux [HBC07]

L'algorithme de Hébert, Bretto et Crémillieux, également appelé MT-Miner (voir Algorithme 9), rappelle APriori [AS+94], un algorithme générique pour résoudre des problèmes d'énumérations. L'idée est d'engendrer l'ensemble des solutions en effectuant un parcours en largeur d'un espace de candidats, de solutions potentielles. L'espace des candidats est exactement l'ensemble des irredondants.

Définition 5.9 *Irredondant*

Soit un hypergraphe $\mathcal{H} = (V, E)$. Un *irredondant* est un ensemble de sommets $I \subseteq V$ telle que

$$\forall i \in I, \exists H \in E, I \cap H = \{i\}$$

Si une traverse est irredondante alors elle est minimale.

Dans l'exemple 5.1, l'ensemble $\{2, 5\}$ est irredondant mais ce n'est pas une traverse car il n'intersecte pas l'hyperarête $\{1, 3, 4\}$.

Algorithme 9 : Algorithme de Hébert, Bretto et Crémillieux

Entrées : un hypergraphe $\mathcal{H} = (V, E)$
Sorties : l'ensemble des traverses minimales de $\mathcal{H}$
$Res \leftarrow \{\{v\} \mid \{v\}$ est une traverse de $\mathcal{H}\}$;
$C_1 \leftarrow \{\{v\} \mid v \in V \setminus Res\}$;
$j \leftarrow 1$;
tant que $C_j \neq \emptyset$ **faire**
  $C_{j+1} \leftarrow \emptyset$;
  pour tous les $(V_1, V_2) \in C_j \times C_j$ *tel que* $|V_1 \cap V_2| = j - 1$ **faire**
    $Candidat \leftarrow V_1 \cup V_2$;
    si $Candidat$ *est irredondant* **alors**
      si $Candidat$ *est une traverse* **alors**
        $Res \leftarrow Res \cup Candidat$;
      sinon
        $C_{j+1} \leftarrow C_{j+1} \cup Candidat$;
  $j \leftarrow j + 1$;
retourner Res ;

La complexité de l'algorithme est bornée par le nombre de candidats à traiter, c'est à dire par le nombre d'irredondants, multiplié par un polynôme (je reste flou à dessin, car le coût est dépendant de l'implantation). Si la boucle $PourTout$ peut paraître coûteuse, il est en réalité uniquement nécessaire de trouver les ensembles V_1 et V_2 de cardinal j partageant un préfixe V de taille $j - 1$. Il est donc possible de représenter les candidats comme stockés dans un trie : les couples (V_1, V_2) sont des feuilles partageant un même noeud parent. Il est possible de vérifier en temps au plus $\mathcal{O}(mj)$ si un candidat est une traverse. De même, ajouter un candidat dans Res ou C_{j+1} dépend bien sûr de l'implantation de l'algorithme, mais peut se faire en temps linéaire dans la taille du Candidat si l'on utilise des tries.

Dans le pire des cas, l'algorithme n'est pas polynomial dans la taille de l'entrée et de la sortie, comme le montre Thomas Hagen en utilisant les hypergraphes de Takata [Hag08]. Soit N la somme des tailles de l'entrée et de la sortie, l'auteur obtient la borne inférieure $N^{\Omega \log \log(N)}$ sur la complexité de l'algorithme.

2.3 Autres algorithmes (liste non exhaustive)

- **[Dong et Li [DL05]]** Il s'agit d'une amélioration de l'algorithme de Berge. L'idée est à chaque ajout d'une hyperarête H, on reporte l'ensemble des solutions de l'itération précédente qui intersecte H dans l'ensemble des nouvelle solutions. On termine ensuite l'itération en appliquant l'algorithme de Berge, mais en triant les solutions restantes de l'itération précédente dans l'ordre croissant. On évite ainsi d'ajouter un ensemble dans les solutions intermédiaires pour l'en retirer ensuite. L'algorithme est très efficace expérimentalement sur des hypergraphes contenant peu d'hyperarêtes, lesquelles sont de petites tailles.
- **[Bailey, Manoukian et Ramamohanarao [BMR03]]** Il s'agit d'une autre approche incrémentale excepté que, contrairement à l'algorithme de Berge, on ajoute les sommets au fur et à mesure et non les hyperarêtes. A noter que l'algorithme utilise Dong et Li pour calculer des étapes intermédiaires sur des hypergraphes dont les hyperarêtes sont "petites".
- **[Kavvadias et Stravopoulos [KS05]]** Cet algorithme parcours l'espace des solutions candidates en profondeur. Il y a donc un gain d'espace car il n'est pas nécessaire de stocker l'ensemble des solutions intermédiaires. L'algorithme utilise également une notion de sommets généralisés, correspondant aux sommets partageant le même voisinage. Cela peut permettre de produire l'hypergraphe transversal sous une forme compressée.
- **[Fredman et Khachiyan [FK96]]** L'algorithme initial résout `THD` dans sa version en logique des propositions : partant d'une formule CNF et d'une formule DNF, l'algorithme tente de produire un contre-exemple. L'algorithme, assez technique, peut-être adapté pour résoudre `THG` : on part d'une formule complète et d'une formule vide, on

appelle l'algorithme qui résout THD afin de produire un contre-exemple, qu'on ajoute dans la formule vide. On recommence ainsi jusqu'à ce que la formule soit la duale de l'autre.

3 Approche probabiliste

Dans cette section, on étudie le nombre moyen de traverses, d'irredondants et enfin de traverses minimales, afin d'obtenir des bornes sur la complexité moyenne que l'algorithme MT-Miner dans les modèles $\mathcal{HG}(m,n,p)$ et $\mathcal{HG}(m,n,\mathbf{v})$. Dans le reste de la section on notera $q = 1-p$ dans le modèle $\mathcal{HG}(m,n,p)$, $\mathbf{v} = (p_1,\dots,p_n)$ et $q_i = 1-p_i$ dans le modèle $\mathcal{HG}(m,n,\mathbf{v})$.

3.1 Le modèle $\mathcal{HG}(m,n,p)$

Soit un ensemble X de sommets de cardinal j. Dans le modèle $\mathcal{HG}(m,n,p)$, la probabilité que celui-ci soit une traverse est :

$$\mathbb{P}(X \text{ est une traverse}) = (1-q^j)^m$$

La Figure 5.7 résume l'ensemble des résultats obtenus grâce à l'étude de cette formule, que l'on détaille ensuite dans les Lemmes 5.10, 5.11 et 5.12.

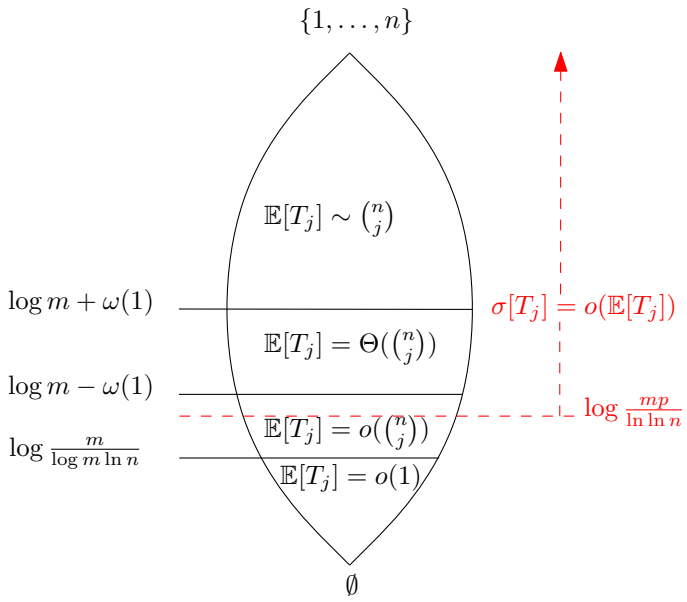


FIG. 5.7 : Cette figure représente le treillis booléen. T_j est l'ensemble des traverses de taille j et $\mathbb{E}[T_j]$ l'espérance du cardinal de cet ensemble. On compare cet espérance au nombre d'ensembles cardinal j. Pour j au dessus d'une certaine taille, on obtient également des résultats sur l'écart-type.

Lemme 5.10 *Nombre moyen de petites traverses*

Soit $j_{min} = \log_{\frac{1}{q}}\left(\frac{m}{\log_{\frac{1}{q}} m \ln n}\right)$. Dans le modèle $HG(m,n,p)$, pour tout $j < j_{min}$, le nombre moyen de traverses $\mathbb{E}[T_j]$ tend vers 0.

Lemme 5.11 *Nombre moyen de grandes traverses*

Dans le modèle $HG(m,n,p)$, on considère j de la forme

$$j = \log_{\frac{1}{q}}(m) + \log_{\frac{1}{q}}(x), \text{ avec } x \in]0,+\infty[$$

1. si $x = \Theta(1)$, une proportion constante de tous les ensembles à j sommets est une traverse. On a $\mathbb{E}[T_j] \sim \binom{n}{j} e^{-\frac{1}{x}}$
2. si x tend vers $+\infty$, presque tout ensemble de taille j est une traverse. On a $\mathbb{E}[T_j] \sim \binom{n}{j}(1-\frac{1}{x})$
3. si x tend vers 0, avec x borné inférieurement par $\frac{1}{m}$, alors presque aucun ensemble de taille j n'est une traverse. On a $\mathbb{E}[T_j] = o\left(\binom{n}{j}\right)$

Les preuves des deux Lemmes 5.10 et 5.11 n'utilisent pas de techniques particulières et sont relativement immédiates. Pour le Lemme 5.11, il suffit de constater que la probabilité d'être un transversal est désormais égale à $\left(1-\frac{1}{mx}\right)^m$ et d'étudier cette fonction pour les différentes valeurs de x. A noter que le cas 3 du Lemme 5.11 recouvre les cas du Lemme 5.10. On entend par "x borné inférieurement par $\frac{1}{m}$" que $x = o(1)$ et $x \leq \frac{1}{m}$.

Nous nous sommes également intéressés à la variance et avons obtenu la formule suivante.

$$\mathbb{V}(T_j) = \sum_{k=1}^{j} \binom{n}{k, j-k, j-k} [(1-2q^j+q^{2j-k})^m - (1-q^j)^{2m}]$$

Une étude de cette fonction nous a permis d'obtenir le résultat suivant.

Lemme 5.12 *Écart-type pour les grandes traverses*

Dans le modèle $HG(m,n,p)$, l'écart-type du nombre de traverses de taille j, pour $j > \log_{\frac{1}{q}} \frac{mp}{\ln n}$ est

$$\sigma[T_j] = \mathcal{O}\left(\mathbb{E}[T_j]\frac{\ln n}{\sqrt{n}}\right)$$

Un ensemble X est un irredondant si pour tout sommet $i \in X$, il existe une hyperarête H telle que $X \cap H = \{i\}$. Pour X de cardinal j fixé, i fixé et H fixé, la probabilité d'un tel évènement est pq^{j-1}. La probabilité qu'un tel i n'existe pas est $1-jpq^{j-1}$. Si X est un irredondant, alors il existe un tuple $(k_1,\ldots,k_j)$ tel que k_i est le nombre d'hyperarêtes H tel que $X \cap H = \{i\}$. La probabilité d'être un irredondant est alors donné par la formule close suivante :

$$\sum_{\forall i \leq j, k_i \geq 1, k_1+\ldots+k_j=\ell j \leq \ell \leq m} \binom{m}{k_1,\ldots,k_j} (pq^{j-1})^\ell (1-jpq^{j-1})^{m-\ell}$$

Théorème 5.13 *Nombre moyen d'irredondants*

Le nombre moyen d'irredondants dans le modèle $HG(m,n,p)$ est

$$\mathcal{O}\left((mn\frac{p}{q^2}\log_{\frac{1}{q}}(\sqrt{nm}))^{\frac{1}{4}(\log_{\frac{1}{q}}(nm) - \log_q p \log_{\frac{1}{q}}(\log_{\frac{1}{q}}(\sqrt{nm})))+1}\right)$$

Il importe d'insister sur le fait que ce théorème donne une borne supérieure sur le nombre moyen d'irredondants. La preuve utilise la majoration suivante du nombre moyen d'irredondants

$$f(j) = \binom{n}{j}\frac{m!}{(m-j)!}(pq^{j-1})^j$$

Cette fonction compte en réalité le nombre moyen d'ensembles X tel que pour chaque sommet i il existe au moins une hyperarête telle que $X \cap H = \{i\}$. On étudie ensuite le ratio $\frac{f(j+1)}{f(j)}$ afin d'obtenir la valeur j qui maximise $f(j)$ et on

obtient

$$j = \lceil \frac{1}{2}(\log_{1/q}(mn) - \log_q(p) - \log_{1/q}(\log_{1/q}(mn))) \rceil$$

Le résultat énoncé par le théorème est donc l'évaluation de f(j) pour cette valeur, multiplié par n. Il est donc évident que cette borne n'est pas fine. Elle permet en revanche d'obtenir le résultat suivant sur la complexité moyenne de l'algorithme MT-Miner.

Théorème 5.14 *Complexité moyenne de MT-Miner*

Dans le modèle $HG(m, n, p)$, il existe une constante c telle que la complexité moyenne de l'algorithme MT-Miner est bornée par

$$\mathcal{O}\left((mn\frac{p}{q^2}\log_{\frac{1}{q}}\sqrt{nm})^{\frac{1}{4}(\log_{\frac{1}{q}} nm - \log_q p \log_{\frac{1}{q}} \log_{\frac{1}{q}} \sqrt{nm})+c}\right)$$

On utilise d'ailleurs la même technique pour obtenir une borne supérieure sur le nombre moyen de traverses minimales borné par la fonction

$$h(j) = \binom{n}{j}\frac{m!}{(m-j)!}(pq^{j-1})^j(1-q^j)^{m-j} = f(j)(1-q^j)^{m-j}$$

Nous avons donc ajouté à $f(j)$ la probabilité que l'ensemble X intersecte les $m - j$ hyperarêtes restantes. Notons donc bien, qu'au moment de l'écriture de ce premier résultat, nous n'avions pas de formule close pour le nombre moyen de traverses minimales.

Théorème 5.15 *Nombre moyen de traverses minimales*

Considérons le modèle aléatoire $HG(m, n, p)$ avec $m = \beta n^\alpha$, $\beta > 0$ et $\alpha > 0$. Il existe une constante positive $c_{\alpha,\beta,p}$, telle que le nombre moyen de traverses minimales est :

$$\mathcal{O}\left(n^{d(\alpha)\log_{\frac{1}{q}}(m)+c_{\alpha,\beta,p}\ln(\ln(m))}\right)$$

avec $d(\alpha) = 1$ si $\alpha \leq 1$ et $d(\alpha) = \frac{(\alpha+1)^2}{4\alpha}$ sinon.

La preuve consiste à étudier le ratio $\frac{h(j+1)}{h(j)}$ en fonction de la valeur de α. On montre que $\sum_{j=1}^{n} h(j) = \mathcal{O}(h(j_0)\ln\ln\ln m)$, où $j_0 = \log_{\frac{1}{q}}(\frac{m(1-\alpha)}{p\ln(m)})$ maximise $h(j)$. Insistons sur le fait que j_0 maximise $h(j)$, soit une borne supérieure du nombre moyen de traverses minimales de taille j et nombre le nombre moyen lui-même. Enfin, on relit le nombre de traverses minimales au nombre de traverses d'une taille donnée.

Lemme 5.16

Considérons ϵ avec $0 < \epsilon < 1$. Dans le modèle $HG(m, n, p)$, le nombre MT de traverses minimales satisfait l'inégalité suivante :

$$\mathbb{P}(MT < \epsilon\mathbb{E}[T_l]) = \mathcal{O}\left(\frac{\ln^2 mn}{(1-\epsilon)^2 n}\right)$$

où T_l est l'ensemble des traverses de taille $l = \log_{\frac{1}{q}} m - \log_{\frac{1}{q}} \ln n + \log_{\frac{1}{q}} \frac{p}{q}$

La probabilité énoncée par le Lemme 5.16 tend vers 0, on sait donc que presque sûrement, le nombre de traverses minimales est supérieure au nombre de traverses annoncé. On obtient donc le résultat suivant.

Théorème 5.17 *Nombre de traverses minimales*

Dans le modèle aléatoire $HG(m, n, p)$, le nombre moyen de traverses minimales est presque sûrement supérieur à

$$n^{\log_{\frac{1}{q}} m + \mathcal{O}(\ln\ln m)}$$

3.2 Le modèle $\mathcal{HG}(m, n, \mathbf{v})$

Ce modèle contenant un très grand nombre de paramètres, il est difficile d'obtenir des résultats avec l'approche probabiliste sans s'imposer quelques restrictions. Dans cette première étude, nous avons choisi de partitionner l'ensemble des sommets en 3 sous-ensembles :

- l'ensemble U des sommets ubiquitaires. Comme leur nom l'indique, ces sommets apparaissent dans presque toutes les hyperarêtes. Soit x une constante. Pour tout sommet $u \in U$, on a $q_u < \frac{x}{m}$ avec $q_u = 1 - p_u$.
- l'ensemble R des sommets rares, qui n'apparaissent que dans très peu d'hyperarêtes. Pour tous sommets $r \in R$, on pose $p_r < 1 - e^{-\frac{1}{\ln n}}$.
- l'ensemble O des autres sommets, tel que $O = V \setminus (U \cup R)$.

Les lemmes suivants sont une conséquence du Théorème 5.13.

Lemme 5.18 *Sommets abondants*

Dans le modèle $HG(m, n, \mathbf{v})$, si l'ensemble des sommets est inclus dans $O \cup U$, alors le nombre moyen d'irredondants est

$$\mathcal{O}\left((mn\ln\sqrt{nm})^{\frac{1}{4}\ln n(\ln nm - 2\ln\ln n - \ln\ln\sqrt{mn})}\right)$$

Lemme 5.19 *Quand la rareté est chose rare*

Dans le modèle $HG(m, n, \mathbf{v})$, si $|R| = \mathcal{O}((\ln n)^c)$, où c est une constante, alors le nombre d'irredondants est quasi polynomial.

Lemme 5.20 *L'influence des sommets ubiquitaires*

Dans le modèle $HG(m, n, \mathbf{v})$, la probabilité d'avoir un nombre polynomial d'irredondants contenant des sommets ubiquitaires tend vers 1.

Les bornes supérieures obtenues sur les irredondants valant pour les traverses minimales, on obtient ainsi le théorème suivant :

Théorème 5.21

Dans le modèle $HG(m, n, \mathbf{v})$, on note M le nombre de traverses minimales.

- Si $|O \cup R| \leq \mathcal{O}(\ln n)$, alors $\mathbb{E}[M]$ est au plus polynomial.
- Si $|R| = \mathcal{O}(\ln n^c)$, où c est une constante, alors $\mathbb{E}[M]$ est au plus quasi-polynomial.
- Si $|R| = \Theta(n)$, alors $\mathbb{E}[M]$ est au plus exponentiel en $|R|$.

L'ensemble des résultats obtenus via l'approche probabiliste nous éclaire sur plusieurs points quant à l'efficacité de l'algorithme MT-Miner.

- Dans les modèles $\mathcal{HG}(m, n, p)$ comme $\mathcal{HG}(m, n, \mathbf{v})$, il existes de nombreuses distributions pour lesquelles l'algorithme est quasi-polynomial en moyenne dans la taille de l'entrée.
- Un paramètre d'importance capitale pour déterminer si l'algorithme va être "efficace" (i.e. polynomial ou quasi-polynomial) est la quantité de sommets rares. Pour reprendre les 4 exemples de la section 1.1, l'algorithme MT-Miner n'a pu terminer que sur le seul l'exemple *mushroom.dat*, c'est à dire le seul n'ayant presque pas de sommets rares. Le calcul des autres exemples a été arrêté après un jour de calcul, saturé la RAM et un disque dur de 500 Go.
- Plus l'hypergraphe possède de sommets rares, plus $Tr(\mathcal{H})$ sera grand.

4 Approche par la combinatoire analytique

Dans cette section, nous avons déployé de nombreux outils de combinatoire analytique pour tenter de résoudre le problème. La Fig. 5.8 résume les différents résultats obtenus.

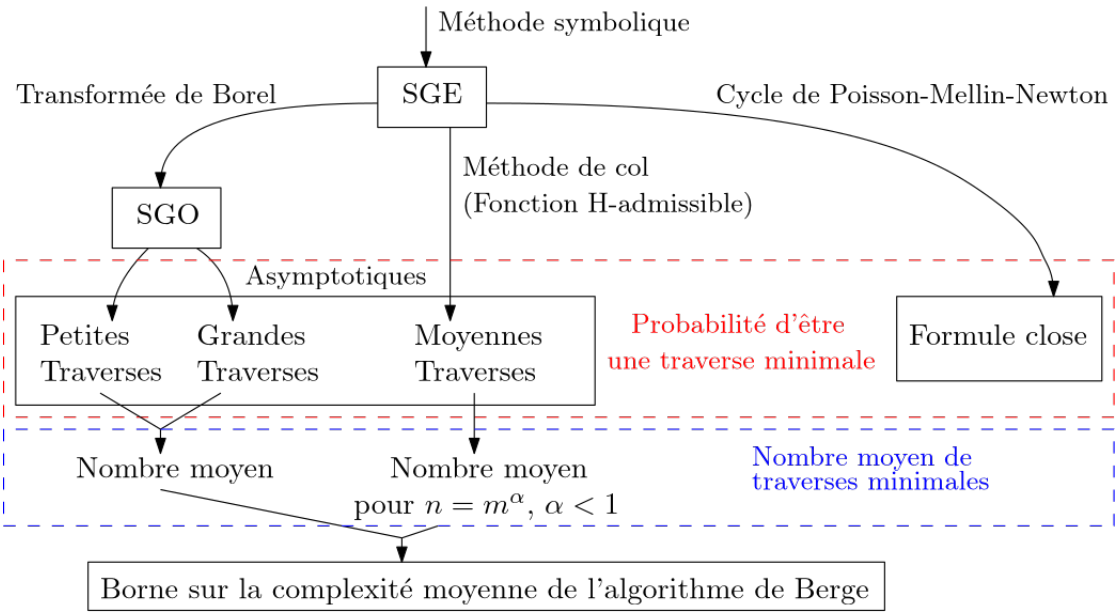


FIG. 5.8 : Cette figure résume les techniques et représentation de la probabilité d'être une traverse minimale utilisées dans cette section. Les différentes représentations ont des interprétations combinatoires distinctes.

4.1 Fonctions génératrices probabilistes

Pour obtenir la série génératrice, on utilise la méthode symbolique et l'argumentaire suivant. Pour qu'un ensemble $C = \{c_1, \dots, c_\ell\} \subseteq \{1, \dots, n\}$ de cardinal ℓ soit une traverse minimale, la matrice de l'hypergraphe associé doit être décomposables en deux parties.

1. Une partie constituée d'au moins ℓ lignes, qui garantit que l'ensemble est un irredondant. Autrement dit, pour chaque colonne $c_i \in C$, il existe au moins une ligne qui contient un 1 à la colonne c_i et uniquement des 0 sur les $\ell - 1$ colonnes restantes de C.
2. Une partie constituée d'au plus $n - \ell$ lignes, qui garantit que l'ensemble est une traverse. Pour cela, chaque ligne doit contenir au moins un 1 parmi les colonnes de C.

Soit un vecteur $\mathbf{v} = (\mathbf{v}_1, \dots, \mathbf{v}_n)$. On considère ℓ sommets de la matrice d'incidence $M(\mathsf{H})$.

Soit l'ensemble des colonnes sélectionnées. On pose $(p_1, \dots, p_\ell) = (\mathbf{v}_{c_1}, \dots, \mathbf{v}_{c_\ell})$ la restriction de $\mathbf{v}$ sur ces ℓ colonnes. La probabilité qu'une ligne de la matrice contienne uniquement des 0 sur ces ℓ colonnes est données par $z_\ell(\mathbf{v}) = (1-p_1)(1-p_2)\cdots(1-p_\ell)$ (on notera z_ℓ au lieu de $z_\ell(\mathbf{v})$ lorsqu'il n'y a pas d'ambiguïté). La probabilité qu'une ligne contienne exactement un 1, à la colonne i, parmi les ℓ colonnes est donnée par $o_{\ell,i} = \frac{p_i}{1-p_i} z_\ell$. Enfin, la probabilité qu'une ligne contienne au moins deux 1 est $1 - z_\ell - \sum_{i=1}^{\ell} o_{\ell,i}$. La `p.g.f.` d'un ensemble non-vide de lignes contenant exactement un 1 à la colonne i est $e^{o_{\ell,i}\cdot u} - 1$. Pour que l'ensemble C soit un irredondant, il faut que cet ensemble existe pour chacun des ℓ sommets de C. Pour que l'ensemble C soit une traverse, l'ensemble des lignes restantes doit contenir au moins deux 1. La `p.g.f.` associée est $e^{o_{\ell,i}\cdot u} - 1$. En combinant ces séries et après une étape de simplification, on obtient le lemme suivant.

Lemme 5.22

Dans le modèle $\mathcal{HG}(m, n, \mu)$, la fonction génératrice probabiliste d'un ensemble C de ℓ sommets d'être une traverse minimale est :

$$S_\ell(u) = \int_{[0,1]^\ell} e^{(1-z_\ell)\cdot u} \prod_{i=1}^{\ell} \left(1 - e^{-o_{\ell,i}\cdot u}\right) d\mu_{\otimes \ell}(p_1, \dots, p_\ell)$$

Un corollaire du Lemme 5.22 est que, dans le modèle $\mathcal{HG}(m, n, \mathbf{v})$, pour un vecteur $\mathbf{v}$ donnée, la fonction $S_\ell(u)$ est simplifiée en

$$S_\ell(u) = e^{(1-z_\ell)\cdot u} \prod_{i=1}^{\ell} \left(1 - e^{-o_{\ell,i}\cdot u}\right)$$

Dans le modèle $\mathcal{HG}(m,n,p)$, tous les $o_{\ell,i}$ sont égaux et peuvent se noter $o_\ell = p(1-p)^{\ell-1}$. On obtient

$$S_\ell(u) = e^{(1-z_\ell)u}\left(1-e^{-o_\ell \cdot u}\right)^\ell = e^{(1-(1-p)^\ell)u}\left(1-e^{p(1-p)^{\ell-1}\cdot u}\right)^\ell$$

La probabilité qu'un ensemble de ℓ sommets soit une traverse minimale d'un hypergraphe à m arêtes est donc donnée par $m![u^m]S_\ell(u)$, où $[u^m]S_\ell(u)$ représente le m-ème coefficient de la fonction $S_\ell(u)$. La fonction est holomorphe, donc analytique.

La transformée de Borel permet de passer d'une série génératrice exponentielle $S_\ell(u)$ à une série génératrice ordinaire $\hat{S}_\ell(u)$. La transformée est définie comme suit :

$$\hat{S}_\ell(u) = \int_0^\infty S_\ell(tu)e^{-t}dt$$

En appliquant la transformée de Borel sur notre série génératrice exponentielle, on obtient le résultat suivant :

$$\hat{S}_\ell(u) = u^\ell \left(\prod_{j=1}^{\ell} o_{\ell,j}\right) \ell! \prod_{i=1}^{\ell} \frac{1}{1-u(1-z_\ell - io_\ell)}$$

Cette série a une interprétation combinatoire différente de la précédente. Rappelons que chaque variable u marque une ligne dans la matrice. Les $u^\ell \prod_{j=1}^{\ell} o_{\ell,j}$ marquent les ℓ lignes qui contiennent exactement un 1. Le $\ell!$ permettant de choisir l'ordre dans lequel ces lignes apparaissent. Appelons ces lignes *séparatrices*. Elles garantissent la propriété de minimalité, ou autrement dit, que l'ensemble est un irredondant. Après que la i-ème séparatrice ait été ajoutée dans la matrice et jusqu'à ce que la prochaine séparatrice soit ajoutée, chaque ligne ne peut contenir que des zéros et ne peut contenir exactement un 1 dans l'une des i positions restantes. Cette probabilité est donnée par $(1-z_\ell - io_\ell)$. Une séquence, ou un bloc, de ces lignes est donnée par le quasi-inverse $\frac{1}{1-u(1-z_\ell-io_\ell)}$. En réunissant les différents blocs entre chaque séparateur, on obtient le produit. Notons que dans le modèle $\mathcal{HG}(m,n,p)$, la fonction se simplifie en : $\hat{S}_\ell(u) = u^\ell o_\ell^\ell \ell! \prod_{i=1}^{\ell} \frac{1}{1-u(1-z_\ell-io_\ell)}$.

4.2 Le cycle de Poisson-Mellin-Newton

On montre ici comment il est possible, en utilisant le cycle de Poisson-Mellin-Newton, de relier l'approche de la combinatoire analytique à l'approche probabiliste. Cette méthode est décrite dans [FS95] (page 21).

On se restreint au modèle $\mathcal{HG}(m,n,p)$. On rappelle la série génératrice exponentielle

$$S_\ell(u) = e^{(1-(1-p)^\ell)u}\left(1-e^{-p(1-p)^{\ell-1}u}\right)^\ell$$

Soit la série génératrice de Poisson $P_l(u) = e^{-u}S_l(u)$. En appliquant à cette série une transformée de Mellin, on obtient

$$M_l(s) = \sum_{k=0}^{l}(-1)^{l-k}\binom{l}{k}\frac{\Gamma(s)}{(1-p)^{(l-1)s}(1-p+lp-kp)^s}$$

Les coefficients de la série génératrice d'origine sont ensuite donné par l'intégrale de Nørlund-Rice suivante :

$$c_{m,l} = \frac{1}{2\pi i}\int_\Lambda \frac{\Gamma(m+1)}{\Gamma(m+s+1)}M_l(s)ds$$

Après plus étapes de simplification on obtient :

$$\sum_{i=\ell}^{m}\binom{m}{i}l!\left\{{i \atop l}\right\}(1-(1-p)^l - l\cdot p(1-p)^{l-1})^{m-i}\cdot(p(1-p)^{l-1})^i)$$

Cette formule nous permet d'obtenir une nouvelle interprétation combinatoire. Soit un ensemble de cardinal ℓ, sa probabilité d'être une traverse minimale d'un hypergraphe à m hyperarêtes est donnée par :

- il existe au moins ℓ lignes qui contiennent exactement un 1, un par sommet. Si l'on suppose qu'il existe i de cette sorte, alors il existe $\binom{m}{i}$ façon de choisir ces lignes.
- les i lignes sont ensuite distribuées parmi les ℓ sommets. Il s'agit d'une composition d'ensemble de $\{1,\dots,i\}$ en ℓ sous-ensembles. Il en existe $l!\left\{{i \atop l}\right\}$, où $\left\{{i \atop l}\right\}$ sont les nombres de Stirling de seconde espèce.
- La probabilité que chacune des i lignes contienne exactement un 1 est $(p(1-p)^{l-1})^i$.
- La probabilité que le reste des $m-i$ lignes contienne au moins deux 1 est $(1-(1-p)^l - l\cdot p(1-p)^{l-1})^{m-i}$.

Nous n'étions pas parvenus à obtenir cette formule lors de notre étude probabiliste du problème. Il est donc amusant de parvenir à l'obtenir alors que tentions une approche via la combinatoire analytique.

4.3 Résultats principaux

Théorème 5.23

Considérons ϵ avec $0 < \epsilon < 1$. Dans le modèle $\mathcal{HG}(m, n, \mathbf{v})$, la probabilité qu'un ensemble X de cardinal ℓ soit une traverse minimale, notée $c_{m,X}$, est :

$$c_{m,X} = \begin{cases} (1-z_\ell)^m \left(1 + \mathcal{O}(\frac{\log m}{m^{1-\epsilon}})\right) & \text{pour} \quad mz_\ell \geq m^\epsilon, \\ \binom{m}{\ell}\ell!(\prod_{i=1}^{\ell} o_{\ell,i}) \left(1 + \mathcal{O}(\frac{\log m}{m^{1-\epsilon}})\right) & \text{pour} \quad mz_\ell \leq m^{-\epsilon}, \\ \frac{m^m}{e^m}\frac{S_\ell(r)}{r^m}\left(1 + \mathcal{O}\left(m^{-1/3} + \frac{l}{m}\right)\right) & \text{pour} \quad m^{-\epsilon} \leq mz_\ell \leq m^\epsilon \end{cases}$$

Les résultats de ce théorème utilisent les deux séries génératrices. Pour $mz_\ell \geq m^\epsilon$ et $mz_\ell \leq m^{-\epsilon}$, les résultats sont obtenus via la série génératrice ordinaire $\hat{S}_\ell(u)$ et la formule de Cauchy. Pour $m^{-\epsilon} \leq mz_\ell \leq m^\epsilon$, le résultat est obtenu en utilisant la méthode du col sur la série génératrice exponentielle $S_\ell(u)$. Ici, $r \sim \frac{m}{1-z_\ell}$ représente le point de col.

En se restreignant au modèle $HG(m, n, p)$ et en utilisant une fois de plus $q = 1 - p$, on obtient le corollaire suivant.

Corollaire 5.24

Dans le modèle $\mathcal{HG}(m, n, p)$, la probabilité qu'un ensemble de cardinal ℓ soit une traverse minimale, notée $c_{m,\ell}$ est :

$$c_{m,\ell} = \begin{cases} (1-z_\ell)^m \left(1 + O(\frac{\log m}{m^{1-\epsilon}})\right) & \text{pour} \quad \ell \leq \log_{\frac{1}{q}} m^{1-\epsilon}, \\ \binom{m}{\ell} o_\ell^\ell \ell! \left(1 + O(\frac{\log m}{m^{1-\epsilon}})\right) & \text{pour} \quad \ell \geq \log_{\frac{1}{q}} m^{1+\epsilon}, \\ e^{-mz_\ell}(1-e^{-o_\ell r})^\ell \left(1 + O(\frac{1}{m^{1-2\epsilon}})\right) & \text{pour} \quad \log_{\frac{1}{q}} m^{1-\epsilon} \leq \ell \leq \log_{\frac{1}{q}} m^{1+\epsilon}, \end{cases}$$

Pour la *somme cumulative*, on obtient les résultats suivants à l'aide de la série génératrice ordinaire.

Proposition 5.2

Dans le modèle $\mathcal{HG}(m, n, p)$, on a

$$\bar{c}_{m,\ell} = \begin{cases} \Gamma(\frac{q}{p})\frac{\ell!}{\Gamma(\frac{q}{p}+\ell+1)}\frac{1}{o_\ell}\left(1 + O(\frac{\log m}{m^\epsilon})\right) & \text{pour} \quad \ell \leq \log_{\frac{1}{q}} m^{1-\epsilon}, \\ \binom{m+1}{\ell+1} o_\ell^\ell \ell! \left(1 + O(\frac{\log m}{m^{1-\epsilon}})\right) & \text{pour} \quad \ell \geq \log_{\frac{1}{q}} m^{1+\epsilon} \end{cases}$$

et pour $\log_{\frac{1}{q}} m^{1-\epsilon} \leq \ell \leq \log_{\frac{1}{q}} m^{1+\epsilon}$

$$\ell! o_\ell^\ell \binom{j_0}{\ell} (1 - z_\ell - \ell o_\ell)^{j_0-\ell} \leq \bar{c}_{m,\ell} \leq (m-\ell)\ell! o_\ell^\ell \binom{j_1}{\ell} (1-z_\ell)^{j_1-\ell}$$

avec $j_0 = \min\left(m, \left\lfloor \frac{\ell}{z_\ell + \ell o_\ell} \right\rfloor\right)$, $\quad j_1 = \min\left(m, \left\lfloor \frac{\ell}{z_\ell} \right\rfloor\right)$.

La méthode utilisée pour obtenir les deux premières asymptotiques est similaires à celle utilisée pour extraire les coefficients . La preuve de la dernière inégalité nécessite d'encadrer $\bar{c}_{m,\ell} = \sum_{i=1}^{m-1} c_{i,\ell}$ en identifiant le i pour lequel $c_{i,\ell}$ est maximale.

Soit $av(n,m,\ell)$ le nombre moyen de traverses minimales de taille ℓ dans le modèle $\mathcal{HG}(m,n,p)$. On a :

$$\sum_{\ell=1}^{min\{m,n\}} av(n,m,\ell) = \sum_{\ell=1}^{min\{m,n\}} \binom{n}{\ell} m![u^m]S_\ell(u) = \sum_{\ell=1}^{min\{m,n\}} \binom{n}{\ell} c_{m,\ell}$$

On obtient les asymptotiques lorsque m est plus grand que n, pour des raisons purement techniques.

Proposition 5.3

Dans le modèle $\mathcal{HG}(m,n,p)$, on suppose qu'il existe une constante $0<\alpha<1$ telle que $n=m^\alpha$. Le nombre moyen de traverses minimales est asymptotiquement équivalent à :

$$av(n,m,\lfloor j \rfloor) + av(n,m,\lfloor j+1 \rfloor) + av(n,m,\lfloor j+2 \rfloor)$$

avec $j = \log_{\frac{1}{1-p}}(\frac{mp}{\kappa}) + 1$, où κ est l'unique solution de l'équation $\alpha + \log_{\frac{1}{1-p}}(\frac{1-e^{-(1-p)\kappa}}{1-e^{-\kappa}}) = 0$

On remarque que, supposer que $m>n$ et $\alpha<1$ est cohérent avec les observations que nous avons effectuées dans les hypergraphes du *Frequent Itemset Mining Dataset Repository*.

L'asymptotique est donc concentrée autour de 3 positions. On montre tout d'abord que $av(n,m,\lfloor j+1 \rfloor)$ est maximale en étudiant le ratio de deux éléments consécutifs. Ce ratio décroît et est proche de 1 lorsque ℓ est proche de j. De plus, il tend vers 0 lorsque $\ell > j+1$ et vers l'infini lorsque $\ell < j-1$, ce qui implique que les autres éléments sont négligeables.

En utilisant la Proposition 5.2, on considère la décomposition suivante :

$$\sum_{i=1}^{m-1} \mathbb{E}_{\mathbf{v}}[Tr_{i,n}] = \sum_{\ell=1}^{\log_{\frac{1}{q}} m^{1-\epsilon}} \binom{n}{\ell} \bar{c}_{m,\ell} + \sum_{\ell=\log_{\frac{1}{q}} m^{1-\epsilon}}^{\log_{\frac{1}{q}} m^{1+\epsilon}} \binom{n}{\ell} \bar{c}_{m,\ell} + \sum_{\ell=\log_{\frac{1}{q}} m^{1+\epsilon}}^{min\{n,m\}} \binom{n}{\ell} \bar{c}_{m,\ell}$$

En démontrant que la première et la troisième somme sont équivalentes au premier et au dernier terme de la seconde somme, on conclut que la seconde somme est la seule qui compte asymptotiquement. On obtient ensuite une borne supérieure de la complexité en identifiant le ℓ qui maximise le coefficient $\bar{c}_{m,\ell}$. La condition sur α est principalement technique.

Théorème 5.25

Dans le modèle $\mathcal{HG}(m,n,p)$, on suppose qu'il existe une constante $1 - \frac{p}{ln(\frac{1}{1-p})} < \alpha < 1$ telle que $n=m^\alpha$. La complexité moyenne de l'algorithme de Berge est

$$\mathbb{E}_p[B(\mathcal{H})] = \mathcal{O}\left(n^{\ell+2} m^4 \log_{\frac{1}{q}}(m) (1-\alpha)^\ell \frac{1}{m^{\frac{1-\alpha}{p}}} \right)$$

avec $\ell = \log_{\frac{1}{q}}(m) - \log_{\frac{1}{q}}(\ln m) - \log_{\frac{1}{q}}(\frac{1-\alpha}{p}) + o(1)$.

Notons qu'une borne supérieure plus précise pourrait être obtenue : un ratio $\frac{\ln(m)^{\ell-1}}{l!}$ a été majoré par m afin d'améliorer la lisibilité du résultat. En effet, on peut voir que le terme dominant est quasi-polynomial en n. En utilisant la Proposition 5.3, on peut démontrer que le nombre de solutions calculées par l'algorithme de Berge est quasi-polynomial. On obtient donc le résultat suivant.

Corollaire 5.26

Dans le modèle $\mathcal{HG}(m,n,p)$, on suppose qu'il existe une constante $1 - \frac{p}{ln(\frac{1}{1-p})} < \alpha < 1$ telle que $n=m^\alpha$. La complexité moyenne de l'algorithme de Berge est quasi-linéaire dans la taille de sa sortie.

Dans nos perspectives, nous souhaitons obtenir le nombre moyen de traverses minimales et la complexité moyenne de l'algorithme de Berge dans le modèle $\mathcal{HG}(m,n,\mu)$ en considérant les mesures suivantes :

- la distribution uniforme entre deux valeurs a et b, c'est à dire $\mu = \mathcal{U}_{[a,b]}$,
- une mesure qui combine deux mesures de Dirac,
- la mesure de Dirac, lorsque la probabilité p est proche de 0.

Notons que, dans le modèle $\mathcal{HG}(m, n, p)$, soit dans le cas de la mesure de Dirac, nous prétendons pouvoir extraire le nombre moyen de traverses minimales lorsque $m < n$. En revanche, pour des raisons purement techniques, nous ne savons pas traiter le cas où $n = m$. Nos résultats pourraient s'appliquer à d'autres algorithmes mentionnés précédemment, comme celui de Dong et Li. Notamment, afin de mieux comprendre pourquoi l'algorithme de Berge est plus efficace en pratique, que celui de Fredman and Khachiyan [FK96], il conviendrait d'analyser cet algorithme en moyenne selon nos différents modèles. Enfin, dans un article récemment déposé sur arxiv [Mar24], Arnaud Mary affirme avoir obtenu un algorithme d'énumération des traverses minimales dont la complexité est bornée par N^{vc}, où vc est la dimension VC de l'hypergraphe. Ce résultat généralise la plupart des cas polynomiaux connus, puisque ces derniers considèrent des classes d'hypergraphes où la dimension VC est bornée par une constante. Dans le cas général, la dimension VC est au plus logarithmique. Cet algorithme pourrait donc détrôner celui de Fredman and Khachiyan, devenant l'algorithme le plus efficace dans le pire des cas. Il serait donc très intéressant d'étudier la dimension VC des hypergraphes aléatoires, afin d'en déduire la complexité moyenne de son algorithme.

6

Analyse en moyenne de l'algorithme naïf de recherche de facteurs

Article(s) associé(s). • *Julien David, Izabell Izkandar, A Study of the Influence of the Entropy on the Behaviour of String Matching Algorithms, soumis à ISAAC 2024.*

Izabell Izkandar a effectué avec moi un stage de 2 mois alors qu'elle était en dernière année de Licence.

Ce chapitre peut être vu comme la suite du chapitre 3. Les ensembles de distributions considérés y sont donc finis. On développe ici l'idée de faire de l'analyse d'algorithmes en moyenne pour des distributions de probabilités qui ne sont pas nécessairement la distribution uniforme. Supposons que l'on souhaite étudier l'influence d'un paramètre t sur le comportement moyen d'un algorithme. Soit $D_{k,t}$ un ensemble fini de distributions à k évènements (ici k représente la taille de l'alphabet sur lequel les mots seront définis) dont l'image par une fonction donnée est égale à t (voir Définition 3.9 dans le chapitre 3 pour plus de précision). Un mot aléatoire est ici produit par une source sans mémoire telle que chaque lettre est tirée aléatoirement selon une distribution $\pi \in D_{k,t}$. Soit $D_{n,k,t}$ l'ensemble des distributions possibles sur les mots aléatoires de taille n définis sur un alphabet de taille k. $D_{n,k,t}$ est l'ensemble des distributions produits :

$$\{\pi_{\otimes n} \mid \pi \in D_{k,t}\}$$

Pour simplifier la notation, on utilisera $D_{n,t}$ au lieu de $D_{n,k,t}$.

La complexité moyenne que l'on souhaite calculer est la suivante :

$$\frac{1}{|D_{n,t}|} \sum_{\pi \in D_{n,t}} CoutMoyen(\pi)$$

Plus un paramètre est important, plus la variance de $CoutMoyen$ est proche de 0 sur $D_{n,t}$. On verra dans ce chapitre que l'on peut même trouver des paramètres tels que $\forall \sigma, \pi \in D_{n,t}$, on a $CoutMoyen(\pi) = CoutMoyen(\sigma)$. On a alors un *paramètre déterminant* permettant de comprendre le comportement d'un algorithme et il suffit d'étudier $CoutMoyen(\pi)$ pour n'importe quel $\pi \in D_{n,t}$.

Dans ce chapitre, on s'intéresse aux algorithmes de recherche de motifs dans du texte, plus précisément à la notion de facteur.

Définition 6.1

Soient les mots u et v définis sur un alphabet Σ, de longueurs respectives m et n. On dit que u est un **facteur** de v si

$$\exists i \in [n-m+1], \forall j \in [m], u_j = v_{i+j-1}$$

Les algorithmes de recherche de facteurs représentent un domaine de recherche actif depuis cinquante ans. Dans [FL10], Thierry Lecroq et Simone Faro analysent expérimentalement 85 algorithmes (le nombre d'algorithmes existants a augmenté depuis) qui comptent le nombre d'occurrences d'un facteur de longueur m dans un texte de longueur n. D'après Thierry Lecroq, le nombre d'algorithmes a aujourd'hui dépassé la centaine. Les auteurs fournissent une cartographie des performances des meilleurs algorithmes, en fonction de la taille du motif ou de l'alphabet. Ils utilisent pour leurs expériences des générateurs aléatoires uniformes, des séquences de génomes ou de protéines, ainsi que des textes en langages naturels. Récemment, différents travaux se sont concentrés sur des méthodes pour comparer l'efficacité des algorithmes [DT17 ; Did19]. Les résultats présentés dans ce chapitre se démarquent de ces résultats et présentent un nouveau point de vue en utilisant le générateur du Chapitre 3.

Comme il est montré dans [THG16], l'algorithme dit "naïf" (voir ci-dessous), implanté avec des instructions SIMD, demeure efficace. Approfondir l'analyse de cet algorithme n'a donc rien d'anachronique.

Dans ce chapitre, on considère qu'une chaîne aléatoire est une séquence de variables indépendantes (issues d'un alphabet à k lettres), distribuées selon une distribution d'entropie (de Shannon ou de collision) fixée t. Notons que, lorsque l'entropie est maximale, la probabilité de tirer une lettre est égale à $\frac{1}{k}$: on obtient un générateur aléatoire uniforme pour un alphabet à k lettres. Plus l'entropie est faible, plus certaines lettres auront une probabilité plus élevée d'apparaître que d'autres. Dans les études qui sont présentées, le texte et le motif sont tous deux aléatoires. Ils peuvent avoir été engendrés par la même source, ou potentiellement par deux sources de même entropie, selon le résultat énoncé.

La section 1 présente l'algorithme naïf de recherche de séquence. On présente des résultats expérimentaux sur l'algorithme naïf dans la sous-section 1.1, obtenus à l'aide du générateur du chapitre 3. Les deux sous-sections suivantes présentent des résultats théoriques sur la complexité moyenne de l'algorithme naïf, exprimée en fonction de l'entropie de Shannon et de l'entropie de collision de la distribution sur les entrées de l'algorithme. On en déduit que l'entropie de collision est un paramètre déterminant, qu'il convient de considérer pour la suite de l'analyse. On montre ensuite que les erreurs de prédictions de branchement jouent un rôle dans l'efficacité de l'algorithme naïf et on étudie l'évolution de ces dernières en fonction de l'entropie de collision. La section 2 se concentre sur d'autres algorithmes de recherche de séquence et vise à montrer via des résultats expérimentaux et théoriques que l'entropie de collision reste un paramètre déterminant.

1 L'algorithme naïf

Algorithme 10 : Algorithme Naïf

Entrées : Un texte v de longueur n, un mot u de longueur m
Sorties : Nombre d'occurrences de u dans v
$occ \leftarrow 0$;
pour $i \in \{1, \cdots, n-m+1\}$ **faire**
  $j \leftarrow 1$;
  tant que $v_{i+j-1} = u_j$ *and* $j \leq m$ **faire**
    $j \leftarrow j+1$;
  si $j = m+1$ **alors**
    $occ \leftarrow occ + 1$;
retourner occ

FIG. 6.1 : L'algorithme naïf de recherche de séquence

L'algorithme naïf de recherche de séquence a une complexité de $\Theta(nm)$ dans le pire des cas, $\Theta(n)$ dans le meilleur, avec une constante égale à 1, et $\Theta(n)$ en moyenne lorsque la distribution uniforme est considérée, avec une constante égale à $\frac{k}{k-1}$. La complexité en temps est liée au nombre de comparaisons entre les lettres du motif et celle du texte.

La complexité moyenne de l'algorithme naïf est évaluée selon le nombre de comparaisons effectuées dans sa boucle *while*. L'algorithme reste dans cette boucle tant qu'une comparaison négative n'a pas été effectuée, c'est à dire tant que v_{i+j-1} est égal à u_j, ou tant que m comparaisons positives n'ont pas été effectuées.

La complexité moyenne de l'algorithme est donc donnée par la formule suivante :

$$\sum_{i=1}^{n-m+1} \left(1 + \sum_{j=1}^{m-1} \prod_{\ell=1}^{j} \mathbb{P}(v_{i+\ell-1} = u_\ell) \right)$$

Si le texte et le motif sont engendrés par la même source et selon la distribution π, la probabilité d'avoir une comparaison positive est égale à $\sum_{i \in \{1,\ldots,n\}} \pi_i^2$. On obtient donc la formule suivante.

$$(n-m+1) \left(1 + \sum_{j=1}^{m-1} \left(\sum_{i=1}^{k} \pi_i^2 \right)^j \right)$$

Dans l'expérience suivante, on mesure le nombre moyen de comparaisons effectuées par l'algorithme naïf sur des entrées engendrées aléatoirement à l'aide des générateurs décris dans le chapitre 3.

1.1 Benchmarks

Précisons tout d'abord que, dans ce chapitre, pour tous les résultats expérimentaux qui sont présentés, chaque point de chaque courbe a été calculé en faisant une moyenne sur 10 000 valeurs. L'alphabet sur lequel on

Affirmer que l'entropie est un paramètre majeur pour les algorithmes de recherche de séquence ne surprendra pas les spécialistes du domaine. En revanche, savoir comment l'entropie influence l'efficacité (c'est à dire effectuer une analyse théorique de cette question), obtenir une fonction continue qui prend en entrée l'entropie à la taille de l'entrée et renvoie le coût moyen d'un algorithme pour ces deux paramètres (à savoir $CoutParametre(t, n)$ comme défini dans l'interlude), n'a encore jamais été fait. Dans un premier temps l'entropie que j'avais considérée dans les expériences était l'entropie de Shannon. Le générateur aléatoire conçu dans le Chapitre 3 se révélant capable de gérer l'entropie de collision, j'ai ensuite opté pour ce paramètre, bien plus efficace pour l'analyse des algorithmes de recherche de séquence.

La complexité moyenne dans le cas uniforme est déjà un bon estimateur du comportement de l'algorithme naïf sur des données réelles. Si ce résultat est bien connu et fait partie du folklore, on peut tout de même le vérifier ici. Dans la figure 6.3, on peut notamment voir que, si l'on utilise la distribution uniforme, le taux d'erreur de prédiction du nombre de comparaisons effectuées est au plus de 3% parmi les 4 exemples représentés.

Dans la FIG. 6.3, on mesure l'entropie t de 4 données réelles, puis on effectue les deux expériences suivantes, en faisant varier la taille du texte n et en fixant $m = 100$:

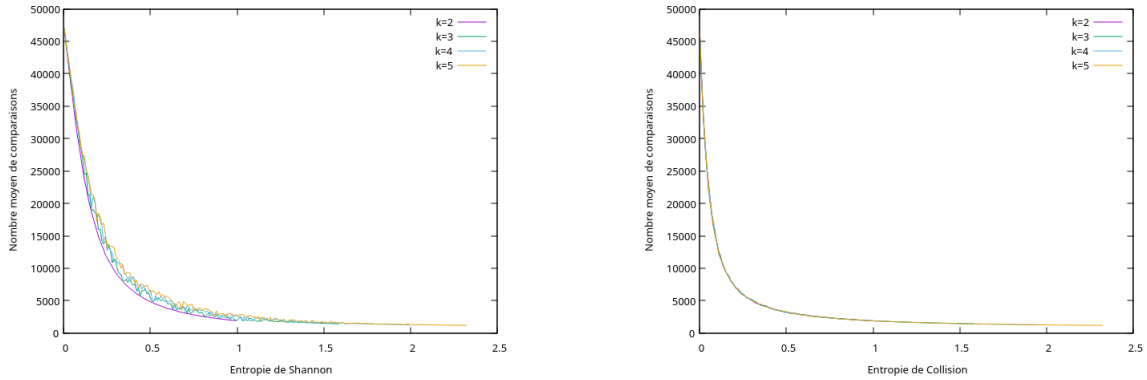


FIG. 6.2 : Évolution de la complexité moyenne de l'algorithme naïf sous des distributions aléatoires dont l'entropie de Shannon (à gauche) et l'entropie de collision (à droite) ont été fixées. Ici, le texte est de longueur 1000, le motif de longueur 50 et ils partagent la même distribution de probabilité. Comme on peut le voir, dans les deux cas, la complexité pire cas est atteinte lorsque les entropies sont proches de 0. La complexité moyenne dans le cas uniforme est atteinte lorsque les entropies sont maximales. Les deux fonctions sont donc *intéressantes*, selon la définition 4.19 de l'interlude. Dans le cas de l'entropie de Shannon, le coût moyen augmente également avec la taille de l'alphabet. L'entropie de collision détermine totalement le coût moyen, c'est une fonction *déterminante*.

1. on extrait aléatoirement un texte de taille n et un motif de m du texte d'origine.
2. on engendre aléatoirement un texte de taille n et un motif de m d'entropie t (ou selon la distribution uniforme, selon la courbe).

Puis dans les deux cas, on mesure le nombre de comparaisons effectuées par l'algorithme naïf pour ce texte et ce motif. On calcule ensuite la différence entre ces des valeurs, que l'on divise par $n \times m$. On appelle cette valeur le *taux d'erreur de prédiction*. Les courbes de la FIG. 6.3 représentent les taux moyen d'erreurs de prédiction, où $10,000$ valeurs on été tirées pour chaque point de la courbe. Comme on peut le voir, les données réelles ont généralement une entropie "suffisamment" élevée. Les prédictions obtenus en supposant que la distribution considérée est la distribution uniforme sont meilleures lorsque l'entropie est proche du maximum. La figure montre que, si l'on considère l'entropie des textes, il est possible d'obtenir de meilleures prédictions du nombre moyen de comparaisons effectuées.

Dans la FIG. 6.3, pour le chromosome 21, les deux prédictions qui utilisent l'entropie donnent un meilleur résultat que celle donnée par la distribution uniforme. Dans le cas des langages naturels, l'entropie de Shannon ne semble pas être un bon paramètre, car la prédiction est moins bonne que celle de la distribution uniforme. En revanche, l'entropie de collision donne toujours un meilleur résultat. Dans le cas de la bactérie, les deux distributions sont très proches de leur valeur maximale, ce qui signifie que les distributions associées sont proches de l'uniforme. Les trois prédictions sont équivalentes et bien plus précises que dans les autres cas.

1.2 Entropie de Shannon

Le théorème suivant se concentre sur la complexité moyenne de l'algorithme naïf lorsque la taille de l'alphabet est égale à 2. Soit $\pi = (x, 1-x)$ une distribution telle que l'entropie de Shannon $H(\pi)$ est égale à une cible t, alors $(1-x, x)$ est la seule autre distribution telle que $H(1-x, x) = t$. C'est donc un cas particulier pour lequel on peut obtenir le résultat suivant.

Théorème 6.2

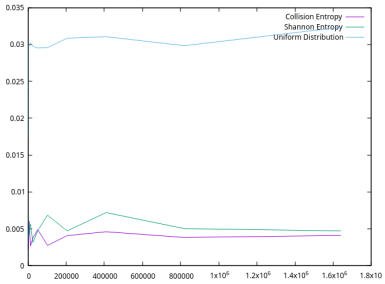
Supposons que $k = 2$ et que le texte et le motif ont été engendrés par des sources d'entropie de Shannon commune t, alors la complexité moyenne de l'algorithme naïf appliqué sur un texte de longueur n et un motif de longueur m est

$$(n-m+1)\left(1 + \frac{y_t - y_t^m}{2(1-y_t)} + \frac{1 - y_t - (1-y_t)^m}{2y_t}\right)$$

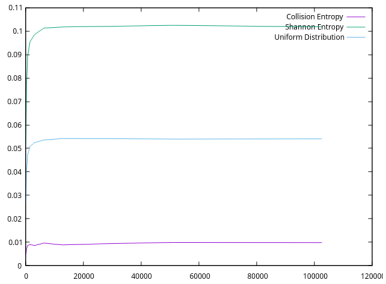
où $y_t = ie(t)^2 + (1-ie(t))^2$ avec ie, pour "inversed entropy", solution de l'équation $H(x, 1-x) - t = 0$.

Dans le cas général, lorsque $k \geq 2$, une formule close peut-être obtenue dans le cas où le texte et le motif partagent la même distribution.

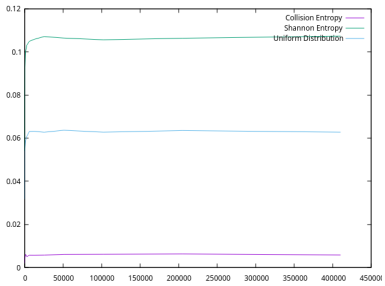
$$(n-m+1)\left(1 + \frac{1}{|D_{n,t}|}\sum_{\pi \in D_{n,t}}\sum_{j=1}^{m-1}\left(\sum_{i=1}^{k}\pi_i^2\right)^j\right)$$



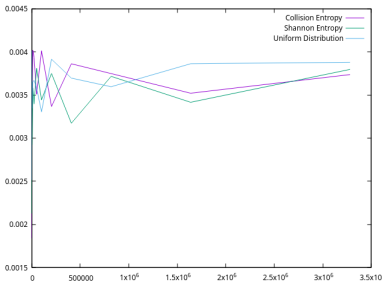
Chromosome 21 ($k = 4$)
Shannon Entropy : 1.9575
Collision Entropy : 1.91818



Hamlet ($k = 88$)
Shannon Entropy : 4.39142
Collision Entropy : 4.23783



Tout du monde en 80 jours ($k = 78$)
Shannon Entropy : 4.34095
Collision Entropy : 3.93015



Ecoli ($k = 4$)
Shannon Entropy : 1.99982
Collision Entropy : 1.99964

FIG. 6.3 : Nous avons effectué une expérience sur 4 textes : un chromosome 21 ($k = 4$), une bactérie "Ecoli" ($k = 4$), "Hamlet" de Shakespeare (version Anglaise), et "Le tour du monde en 80 jours" par Jules Vernes (version Française). Des générateurs aléatoires de distributions à entropie de Shannon et entropie de collision fixées ont également été utilisés, en prenant comme paramètre les entropies mesurées sur les différents textes. Ces figures comparent les différences normalisées d'erreurs de prédiction du nombre de comparaisons effectuées par l'algorithme naïf, c'est à dire la différence normalisée entre le nombre moyen de comparaisons effectuées en utilisant des textes et des motifs aléatoires avec nos générateurs et le nombre moyen de comparaisons effectuées sur les données réelles.

Il est difficile de simplifier ces formules pour en extraire des asymptotiques dans le cas général. C'est en partie dû au fait que l'entropie de Shannon, bien qu'elle soit une fonction *intéressante*, n'est pas une fonction *déterminante* pour l'algorithme naïf.

1.3 Entropie de collision

Pour une entropie de collision t on a $\mathbb{P}_\pi(\text{comparaison positive}) = 2^{-t}$, ce qui nous permet d'obtenir le résultat suivant.

Théorème 6.3

Supposons que $k \geq 2$ et que le texte et le motif ont été engendrés par une même source d'entropie de collision t, alors la complexité moyenne de l'algorithme naïf appliqué sur un texte de longueur n et un motif de longueur m est

$$(n - m + 1)\frac{1 - 2^{-t\times m}}{1 - 2^{-t}}$$

1.4 Temps d'exécution et erreurs de prédictions de branchement

Comme on peut le voir dans la FIG. 6.4, le temps d'exécution de l'algorithme naïf décroît lorsque $t < 0.4$, puis augmente doucement pour $t \in [0.4, 1]$, puis décroît de nouveau. Il n'est donc pas totalement déterminé par le nombre de comparaisons. La différence entre le temps d'exécution observé et le nombre moyen de comparaisons effectué s'explique par les erreurs de prédictions de branchement. L'idée est que le processeur tente de prédire dynamiquement l'issue de chaque test effectué par un programme en fonction des résultats des tests précédents et en utilisant une faible quantité

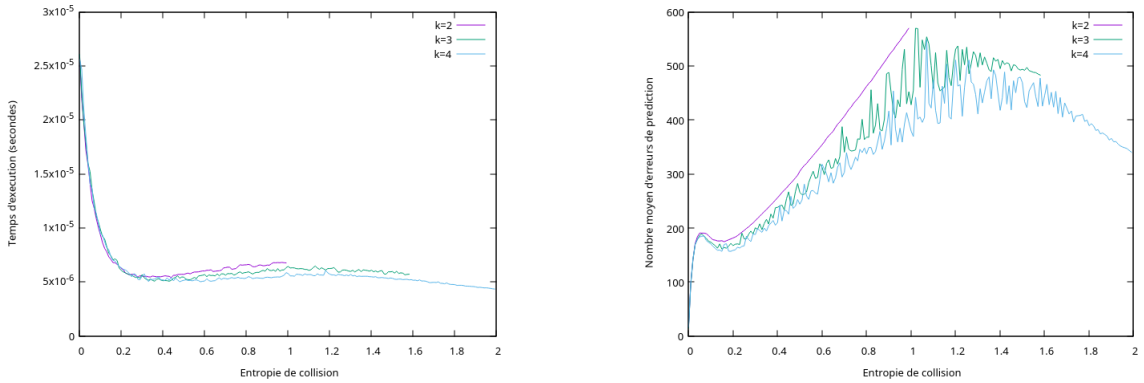


FIG. 6.4 : La figure de gauche représente le temps d'exécution, en secondes, de l'algorithme naïf sur un texte de longueur 1000, un motif de longueur 50 et un alphabet de taille $k \in \{2, 3, 4\}$. La figure de droite montre le nombre d'erreurs de prédictions de branchements moyen effectués lors de l'exécution de l'algorithme.

de mémoire afin de charger le pipeline du processeur avec les instructions qu'il juge le plus probable. Une erreur de prédiction implique de laisser le pipeline du processeur se vider sans enregistrer les résultats des opérations qui n'auraient pas du y être chargées. Plus le pipeline est long, plus cette opération peut être coûteuse en temps.

En utilisant des résultats de [MNW14], on obtient des résultats théoriques sur le nombre moyen d'erreurs de prédictions de branchement de l'algorithme naïf. Un prédicteur dynamique et local à k-bit peut être vu comme une chaîne de Markov à 2^k états, associée à chaque condition du programme, où chaque état indique une prédiction du prochain résultat de ladite condition, en se basant sur les k derniers résultats obtenus.

Dans [MNW14] et [ANP16], les auteurs considèrent différents modèles théoriques de prédicteurs : 1-bit, 2-bit saturating-counter, 2-bit flip-on-consecutive, 3-bit saturating-counter. Ils étudient le nombre moyen d'erreurs de prédictions de branchement dans des algorithmes tels que Quicksort, Quickselect, l'exponentiation et [ANP16] parviennent même à améliorer l'efficacité de Quickselect et de l'exponentiation rapide en réduisant le nombre d'erreurs de prédiction. Nous invitons les personnes désireuses d'en savoir plus sur les prédicteurs de consulter les articles précédemment cités. En particulier, dans [MNW14], les auteurs montrent que, si l'on suppose qu'une condition est vraie avec probabilité p, alors la probabilité d'obtenir une erreur de prédiction de branchement est donnée par les formules suivantes, en fonction du prédicteur local :

$$1bit(p) = 2p(1-p) \qquad 2bit\_sc(p) = \frac{p(1-p)}{1-2p(1-p)}$$
$$2bit\_flip(p) = \frac{2p^2(1-p)^2+p(1-p)}{1-p(1-p)} \qquad 3bit\_sc(p) = \frac{p(1-p)(1-3p(1-p)}{1-2p(1-p)(2-p(1-p)}$$

L'algorithme naïf utilise deux conditions pour lesquelles le processeur peut produire un nombre pertinent d'erreurs de prédictions :

1. la condition $j = m + 1$ à la fin de la boucle *for*
2. la condition de la boucle *while*

Dans la conjecture suivant, la fonction $pred$ peut être remplacée par l'une des fonctions mentionnées précédemment dans l'ensemble $\{1bit, 2bit\_sc, 2bit\_flip, 3bit\_sc\}$.

Conjecture 6.1

Supposons que le texte et le motif soit engendrés par une même source d'entropie de collision t, sur un alphabet à k lettres, avec $k \geq 2$, alors le nombre moyen d'erreurs de prédictions de branchements de

- la condition $j = m + 1$ est $(n - m + 1)pred(2^{-tm})$
- la boucle *while* est supérieur à $(n - m + 1)pred(2^{-t})$

Le nombre moyen d'erreurs donné pour la boucle *while* est bien une borne inférieure. Il correspond aux erreurs obtenues lorsque l'on tente de rentrer pour la première fois dans la boucle. Comme on peut le voir dans la FIG. 6.5, cette approximation grossière donne néanmoins des résultats satisfaisants. En revanche, le fait de le pas considérer tous les tests effectués par la boucle while empêche d'utiliser directement les résultats de [MNW14]. En effet, ceux-ci supposent que les prédicteurs, que l'on peut voir comme des chaînes de Markov, ont déjà atteint la distribution stationnaire. Même si les résultats expérimentaux tendent à dire que la borne inférieure sur la boucle while semble correcte, la chaîne de Markov est en réalité plus complexe que celle décrite par les prédicteurs de [MNW14] et il reste à démontrer qu'elle converge bien vers une distribution stationnaire.

La FIG. 6.5 montre le nombre moyen d'erreurs de prédictions selon les 4 différents modèles de prédicteurs. La différence avec les résultats observés dans la FIG. 6.4 s'explique par le fait que les processeurs disposent en réalité d'informations

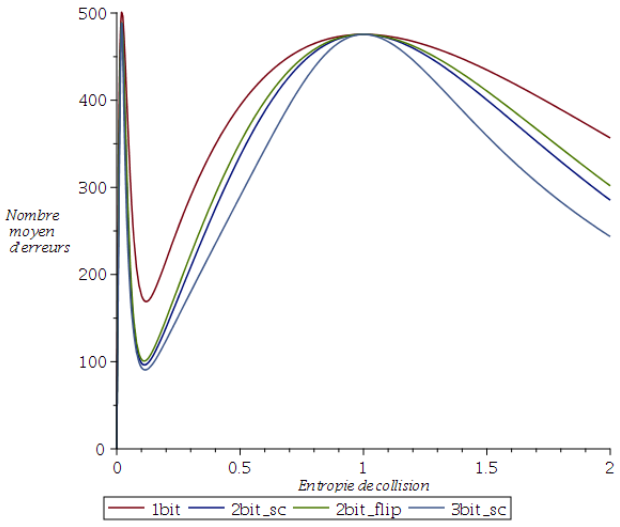


FIG. 6.5 : Dans cette figure, la première partie de la courbe, lorsque $t < 0.2$, correspond à l'estimation théorique du nombre d'erreurs de prédiction sur la condition $j = m + 1$. Comme on peut le voir, ce résultat surestime celui observé en pratique mais le comportement reste similaire. La seconde partie de la courbe, lorsque $t > 0.2$, correspond aux erreurs de prédictions de la boucle *while*. Il s'agit d'une sous-estimation des valeurs observées mais le comportement est de nouveau assez similaire : le nombre d'erreurs croît jusqu'à $t = 1$ puis commence à décroître.

supplémentaires pour faire de la prédiction, comme les prédicteurs globaux qui permettent d'établir des corrélations entre les branchements. Cela pourrait expliquer pourquoi le nombre d'erreurs de la condition $j = m + 1$ est une surestimation : le processeur parvient à mieux prédire le résultat de cette condition grâce aux résultats précédents obtenus sur la boucle *while*.

2 Étude expérimentale d'autres algorithmes

Une question naturelle est ensuite de se demander si notre approche est intéressante pour d'autres algorithmes permettant de résoudre le même problème.

Les figures ci-dessous montrent un échantillon d'algorithmes de recherche de séquences issues de l'outil SMART [Far+16]. Notons que, l'objectif ici n'est pas de découvrir quel est l'algorithme le plus efficace, mais de vérifier l'importance de l'entropie comme clé de compréhension du comportement des algorithmes. Les algorithmes représentés parmi la centaine d'algorithmes existants servent à illustrer les différents comportements. On notera en particulier que certains algorithmes sont conçus pour être efficaces pour un alphabet/texte/motif de petite/grande taille. Par exemple, l'algorithme FJS [FJS07] est connu pour être efficace lorsque la taille de l'alphabet k est supérieure à 15. La FIG. 6.6 montre le nombre de comparaisons moyen effectué entre le texte et le motif. La FIG. 6.7 compare les temps d'exécution des mêmes algorithmes. Un zoom est effectué lorsque l'entropie est supérieure à 0.3 afin de constater que le temps d'exécution de certains algorithmes réaugmente, comme c'était le cas pour l'algorithme naïf. La FIG. 6.8 montre l'évolution des erreurs de prédiction de branchement.

D'après les FIG. 6.6, 6.7 et 6.8, l'efficacité de l'algorithme **KMP** semble plus liée au nombre de prédictions de branchement qu'aux variations du nombre de comparaisons. Contrairement à l'algorithme naïf, la condition $j = m + 1$, également présente dans cet algorithme, ne semble pas créer d'erreur de prédiction de branchements. La borne inférieure utilisée pour l'algorithme naïf sur les erreurs produites par la boucle while reste valide pour **KMP**. On obtient donc le lemme suivant.

Conjecture 6.2

Supposons que le texte et le motif soient engendrés par une même source d'entropie de collision t, sur un alphabet à k lettres, avec $k \geq 2$. Alors le nombre moyen d'erreurs de prédictions de branchement de l'algorithme **KMP** est supérieur à $(n - m + 1)pred(2^{-t})$.

De plus, tous les algorithmes considérés ici, à l'exception de **fsbndm** et **Hash8**, partagent des erreurs de prédictions de branchement avec l'algorithme naïf (voir FIG. 6.8), ce qui suggère que la borne inférieure obtenue dans le Lemme 6.1 fournit déjà une bonne explication théorique à l'augmentation du temps d'exécution des algorithmes de la FIG. 6.7.

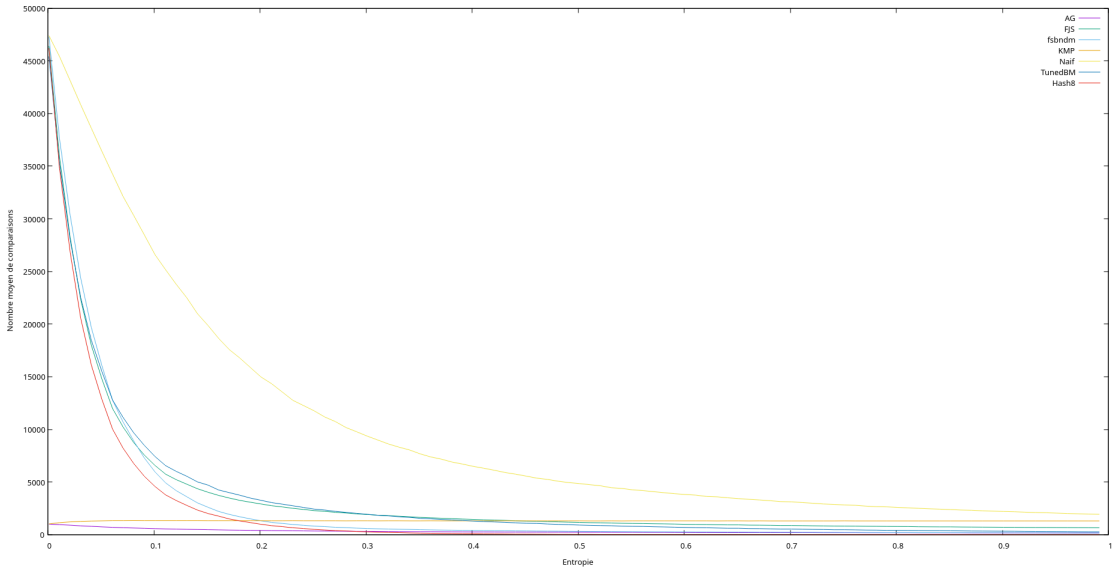


FIG. 6.6 : Nombre moyen de comparaisons pour $n = 1000$, $m = 50$, $k = 2$ et une entropie de Collision qui varie. Comme on peut le voir, l'entropie de Collision est un paramètre essentiel pour la compréhension de leur efficacité. La plupart des algorithmes ont une complexité $\mathcal{O}(nm)$ dans le pire des cas, atteinte lorsque l'entropie est proche de 0, à l'exception de KMP [KMP77] et AG [AG86], dont la complexité pire cas est $O(n)$. Notons également que seules les comparaisons entre les lettres du texte et du motif sont ici comptabilisées. Certains calculs effectués ne sont donc pas considérés dans cette figure, notamment pour Hash8 [Lec07].

3 Conclusion

Ce travail est donc une prémisse de ce que j'aimerais réaliser à l'avenir : partir d'une formule close de la complexité moyenne d'un algorithme et tenter d'en extraire des paramètres déterminants pour le comportement moyen de l'algorithme. De nombreux algorithmes, dans de nombreux domaines, utilisent des boucles *while* dont le test d'arrêt est lié à une comparaison positive ou négative. Il y a donc fort à parier que les entropies de Shannon et de collision jouent un rôle déterminant dans bon nombre d'algorithmes. Sur le long terme, j'aimerais établir une cartographie des algorithmes et des paramètres auxquels la complexité moyenne est sensible.

De plus, l'analyse théorique des erreurs de prédictions de branchement devrait permettre d'obtenir une compréhension plus fine de l'évolution du temps de calcul des algorithmes. Les résultats théoriques existants s'appliquent à des prédicteurs locaux dans lesquels le résultat de chaque test est indépendant des autres. J'aimerai trouver des personnes avec qui collaborer sur ce sujet afin d'étendre les résultats théoriques existants.

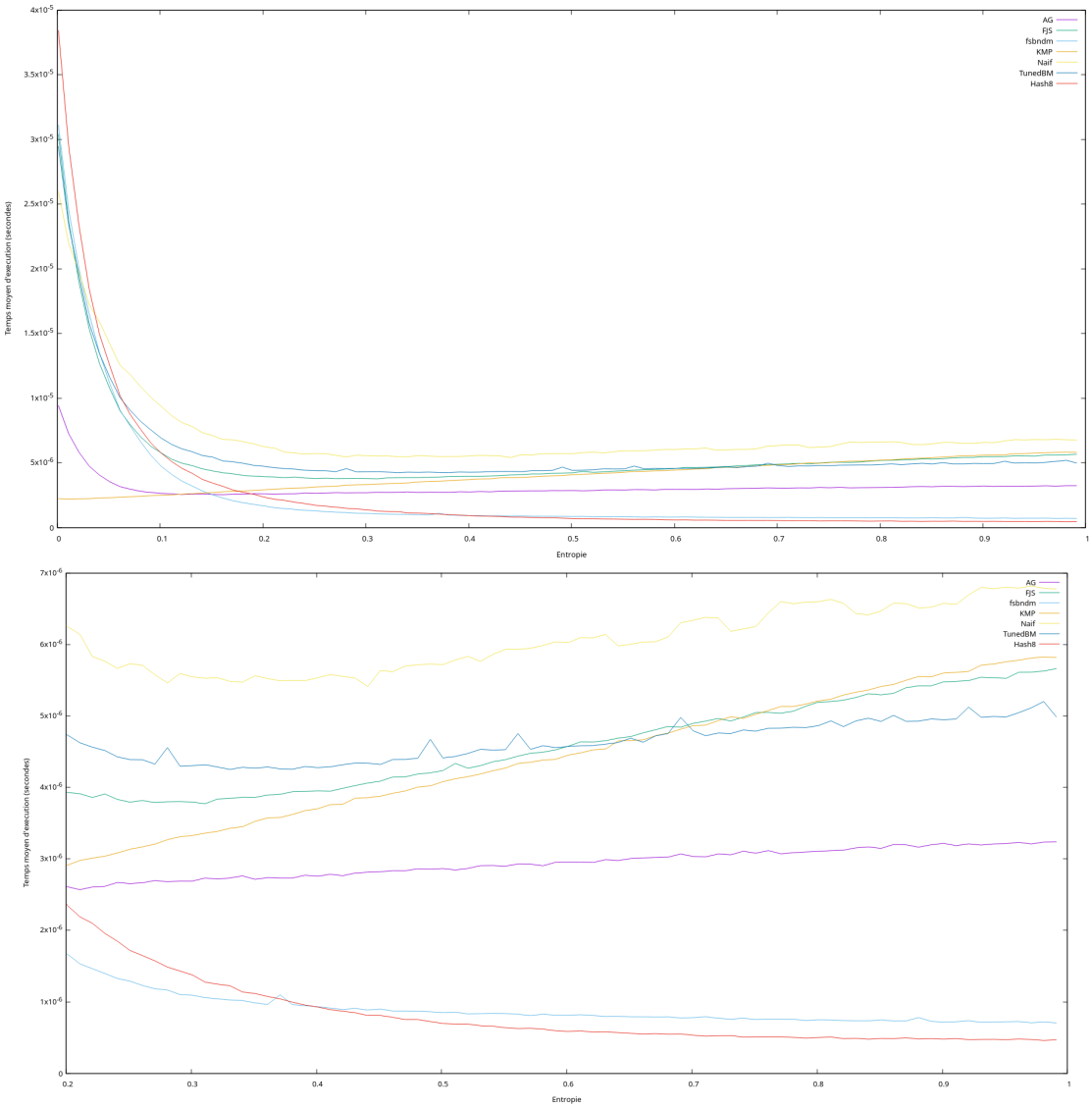


FIG. 6.7 : Temps moyen d'exécution pour $n = 1000$, $m = 50$, $k = 2$. Entropie entre 0 et 1 (en haut) et entre 0.2 et 1 (en bas, effet zoom). Certains algorithmes, comme **fsbndm** [PT11] et **Hash8** [Lec07], sont moins efficaces que l'algorithme naïf lorsque l'entropie est basse, puis deviennent les deux algorithmes les plus efficaces lorsque l'entropie est supérieure à 0.2 (ce qui est le cas dans la plupart des données réelles). Le temps d'exécution de **KMP** [KMP77] augmente jusqu'à dépasser celui de l'algorithme naïf lorsque l'entropie est proche de la valeur maximale.. Hormis **KMP**, **fsbndm** or **Hash8**, le temps moyen d'exécution diminue rapidement puis réaugmente doucement, bien que le nombre moyen de comparaisons soit strictement décroissant. Comme pour l'algorithme naïf, ce phénomène est principalement lié aux erreurs de prédictions de branchement. Lorsque l'entropie augmente, il devient plus difficile pour le processeur de prédire l'issue d'une comparaison.

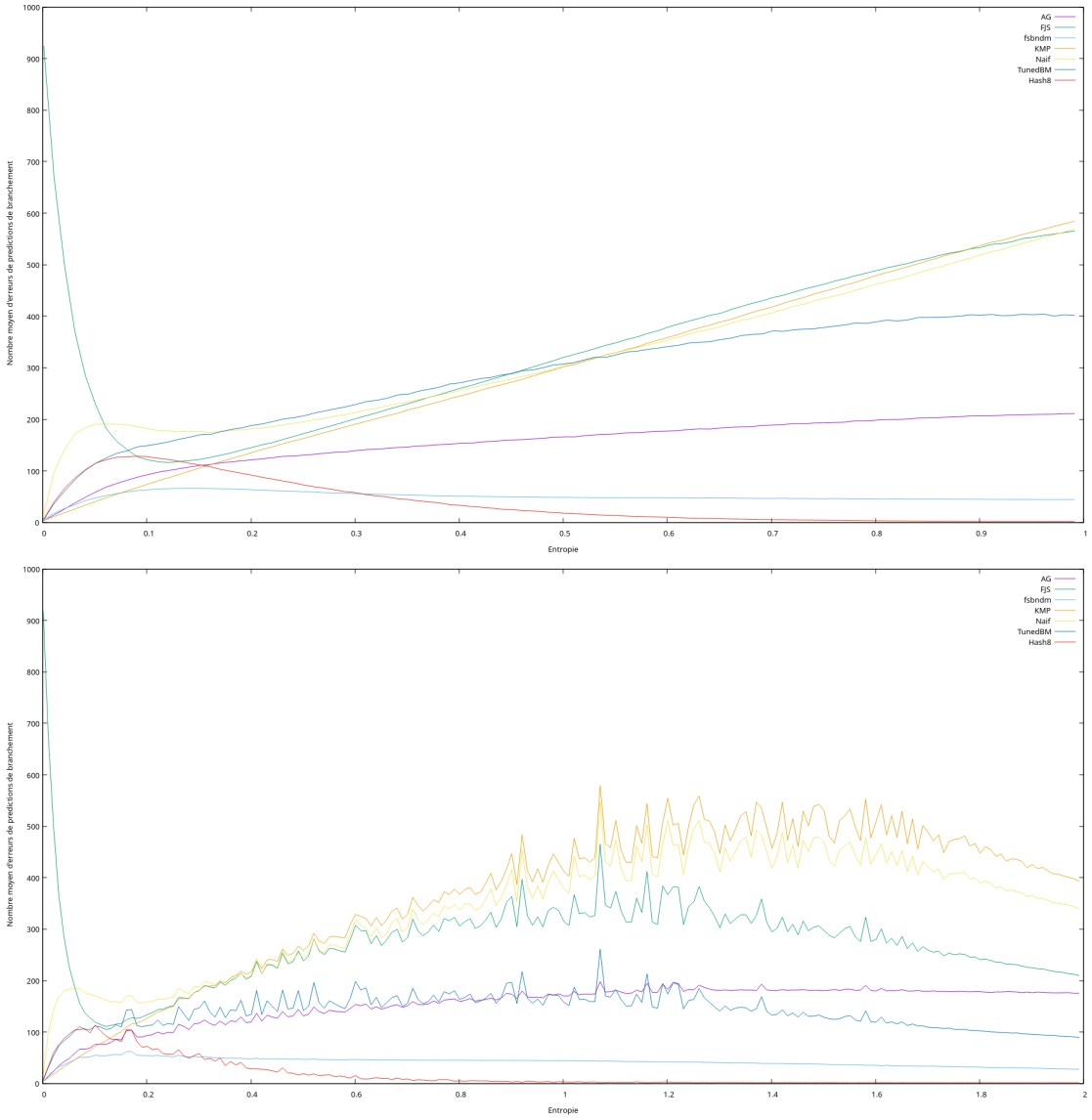


FIG. 6.8 : Nombre moyen d'erreurs de prédiction de branchement pour $n = 1000$, $m = 100$, $k = 2$ (en haut) et $k = 4$ (en bas). Le nombre moyen d'erreurs de prédictions augmente pour t entre 0.2 et 1. pour tous les algorithmes à l'exception de **fsbndm**, où la valeur semble se stabiliser et **Hash8**, ou elle semble tendre vers 0 lorsque l'entropie croît. Pour la figure où $k = 4$, on a effectué peu de pas avec la chaîne de Markov, ce qui explique les graphiques en dents de scies. Ceux-ci sont révélateurs : les pics semblent se produire au même moment pour tous les algorithmes sauf **Hash8** et **fsbndm**. Cela pourrait s'expliquer par le fait que les algorithmes partagent un certain nombre d'erreurs de prédiction de branchement.

7

Recherche approchée dans les séquences ordonnées

Article(s) associé(s). • *Bastien Auvray, Julien David, Richard Groult, Thierry Lecroq. Approximate Cartesian Tree Matching : An Approach Using Swaps. SPIRE 2023 : 49-61*

Bastien Auvray a été encadré par moi-même, Richard Groult et Thierry Lecroq lors de son stage de Master 2. Il commencera en septembre 2024 un doctorant sous notre supervision.

Dans ce chapitre, on considère des séquences d'entiers totalement ordonnées. Par facilité, on pourra considérer sans perte de généralité qu'une séquence de longueur n est une permutation de même taille. Dans le cas où les séquences étudiées ne seraient pas totalement ordonnées, il est possible de considérer pour deux positions de même valeur que la position la plus petite possède la valeur la plus petite, et ainsi réobtenir un ordre total, voire se ramener à une permutation. Pour une séquence z, $|z|$ sera la longueur de z, $z[i]$ le i-ème élément de z et $z[i \dots j]$ le facteur de z débutant au i-ème élément et se terminant au j-ème élément.

Les arbres cartésiens ont été introduits par Vuillemin en 1980 [Vui80]. Principalement associés à des séquences de nombres, ils sont structurés comme des tas à partir desquels les séquences originales peuvent être retrouvées en effectuant un parcours infixe.

Une séquence z de longueur n peut être associée à son arbre cartésien $C(z)$ selon les règles suivantes (voir exemple Fig. 7.1) :

- si z est vide, alors $C(z)$ est l'arbre vide,
- si $z[1 \dots n]$ n'est pas vide et $z[i]$ la plus petite valeur de z, $C(z)$ est l'arbre cartésien enraciné en i, ayant l'arbre cartésien de $z[1 \dots i-1]$ comme sous-arbre gauche et l'arbre cartésien de $z[i+1 \dots n]$ comme sous-arbre droit.

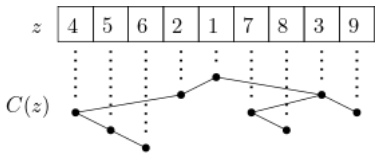


Fig. 7.1 : Une séquence $z = (4, 5, 6, 2, 1, 7, 8, 3, 9)$ et son arbre cartésien $C(z)$.

Leurs liens avec les arbres de Lyndon [CR20], les Range Minimum Queries [DLW14], ou encore les constructions d'arbres suffixes en parallèle [SB14] ont depuis été démontrés.

Récemment, Park *et al.* [Par+19] ont introduit une nouvelle métrique de recherche de motif, appelée *Cartesian Tree Matching* (CTM). Il s'agit de trouver tous les facteurs d'un texte t qui possèdent le même arbre cartésien qu'un motif donné p.

On notera $x \approx_{CT} y$ si deux séquences x et y partagent le même arbre cartésien. Par exemple, on a :

$$z = (4, 5, 6, 2, 1, 7, 8, 3, 9) \approx_{CT} (3, 4, 8, 2, 1, 7, 9, 5, 6)$$

Le problème CTM (Cartesian Tree Matching) de recherche de motif d'arbre cartésien consiste à trouver tous les facteurs d'un texte qui partagent le même arbre cartésien qu'un motif passé en entrée. Il fut défini formellement par Park *et al.* [Par+19] de la façon suivante :

Définition 7.1 *Cartesian tree matching (CTM)*

Soient deux séquences $p[1 \dots m]$ et $t[1 \dots n]$, trouver toutes les positions $1 \leq i \leq n - m + 1$ telles que $t[i \dots i + m - 1] \approx_{CT} p[1 \dots m]$.

CTM peut par exemple être utilisé pour trouver des motifs dans des séries temporelles, comme l'évolution des prix, des données génétiques, des partitions de musique. Le lecteur ou la lectrice pourrait légitimement se demander pourquoi on ne s'intéresse pas simplement à identifier des sous-séquences pour lequel l'ordre relatif des éléments est le même que dans une permutation passée en entrée. Ce problème est appelé *Order-Preserving Matching* [Kim+14 ; Cho+15] et peut être résolu en temps linéaire dans la longueur de la séquence t. CTM est une généralisation de cette notion de motif, puisqu'elle ignore l'ordre relatif entre deux positions séparée par une plus petite valeur.

Park *et al.* ont également introduit une représentation linéaire des arbres cartésiens (voir Section 1), appelée table des distances parentales. Ils donnent des solutions en temps linéaire pour la recherche de motif unique et multiple, en utilisant la représentation sous forme de table des distances parentales et en adaptant des algorithmes classiques comme Knuth-Morris-Pratt [KMP77] (un seul motif) et Aho-Corasick [AC75] (dans le cas où on a plusieurs motifs). Des solutions plus efficaces en pratique ont été donnés dans [Son+21]. De plus, de nouveaux résultats sur CTM sont apparus dans [Par+20 ; KC21 ; Nis+21 ; Far+23]. Il est possible de le résoudre en temps linéaire dans la taille du texte et du motif. L'ensemble de ces résultats concerne la recherche exacte de motifs sur des séquences contiguës de symboles. Le seul résultat connu sur des symboles non contigus se trouve dans un algorithme de recherche d'épisodes [Oiz+22].

Aucun résultat n'étant connu sur la recherche approchée dans le cadre des arbres cartésiens[1], nous avons tenté d'introduire cette notion. En effet, les données réelles sont souvent bruitées. Il est donc important de trouver des facteurs similaires, dans une certaine mesure, à notre motif.

Dans ce chapitre, nous considérons la recherche de motif d'arbre cartésien à un *swap* près, c'est à dire que l'on autorise une transposition de deux symboles adjacents dans la séquence d'origine. Il s'agit d'une perturbation courante dans les données réelles, il semblait donc naturel de les considérer sur les motifs d'arbre cartésien.

La recherche de motif à un swap près, étudiée depuis 1997 [Ami+97], a reçu beaucoup d'attention en recherche de motif dite classique (l'article de Simone Faro [FP18] contient de nombreuses références).

Le chapitre contient la description de deux algorithmes permettant de trouver toutes les occurrences des motifs d'arbres cartésien à un swap près. Ces algorithmes se basent sur une représentation linéaire des arbres cartésiens, définie par Park *et al.* [Par+19] : la *table de distances parentales*. Le premier algorithme a une complexité en temps de $\Theta(mn)$ dans le pire des cas et utilise une caractérisation de tables de distances parentales après un swap. Le second a une complexité en temps de $\mathcal{O}((m^2+n)\log m)$ et utilise un graphe pour engendrer toute les représentations linéaires des arbres cartésiens pouvant être obtenus après avoir effectué un swap sur le motif de départ. Enfin on introduit une représentation linéaire similaire à celle d'Ohlebusch[Ohl13], que nous avons nommée Skipped-Number representation. Nous avons également caractérisé l'effet d'un swap sur cette représentation. Ce résultat n'est pas encore publié mais a ouvert plusieurs pistes de recherche pour obtenir un algorithme efficace.

1 Table des distances parentales

Afin de résoudre CTM sans construire un arbre cartésien à chaque position du texte, une représentation efficace a été introduite par Park *et al.* [Par+19] : la table des distances parentales (voir exemple Fig. 7.2).

Définition 7.2 *Table des distances parentales*

Soit une séquence $x[1\dots n]$, la table des distances parentales x est une séquence d'entiers $\overrightarrow{PD}_x[1\dots n]$, définie comme suit :

$$\overrightarrow{PD}_x[i] = \begin{cases} i - max_{1\leq j<i}\{j \mid x[j] < x[i]\} & \text{si } j \text{ existe} \\ 0 & \text{sinon} \end{cases}$$

La représentation sous forme de table des distances parentales étant en bijection avec les arbres cartésiens, elle peut donc les remplacer sans perte d'information. Notons que, pour un motif p de longueur m, la table $\overrightarrow{PD}_p[1\dots n]$ peut-être calculée en temps et en espace $\Theta(m)$, utilisant une pile d'entiers. Il est également possible de calculer la table des distances parentales des facteurs du texte $t[i\cdots i+m-1]$ *à la volée*, en temps amorti constant. Park *et al.* [Par+19] utilisent ces deux avantages pour décrire un algorithme similaire à celui de Knuth-Morris-Pratt [KMP77], évoqué dans le chapitre précédent. Ils calculent une table des bords de $\overrightarrow{PD}_p$ et l'utilisent pour savoir à tout instant le plus long préfixe commun entre p et $t[i\cdots i+m-1]$. Lorsque ce préfixe est de longueur m, on a trouvé une occurrence du motif.

Par la suite, afin de résoudre efficacement le problème de la recherche approchée que nous allons définir, nous avons introduit une seconde table des distances parentales, qui consiste à parcourir l'arbre de droité à gauche et à trouver le plus proche parent vers la droite, au lieu de le faire vers la gauche comme dans la définition classique. La Fig. 7.2 montre un exemple de ces deux représentations.

Définition 7.3 *Table des distances parentales inverse*

Soit une séquence $x[1\dots n]$, la table des distance parentale inverse de x est une séquence d'entiers $\overleftarrow{PD}_x[1\dots n]$, définie comme suit :

$$\overleftarrow{PD}_x[i] = \begin{cases} min_{i>j\geq n}\{j \mid x[i] > x[j]\} - i & \text{si } j \text{ existe} \\ 0 & \text{sinon} \end{cases}$$

[1] A noter que, pendant l'écriture de ce manuscrit, un article de Sungmin Kim et Yo-Sub Han intitulé *Approximate Cartesian Tree Pattern Matching* et portant sur la recherche approchée de motifs d'arbre Cartesien, où la distance d'édition ou la distance de Hamming ente deux motifs est bornée, a été accepté à la conférence DLT 2024.

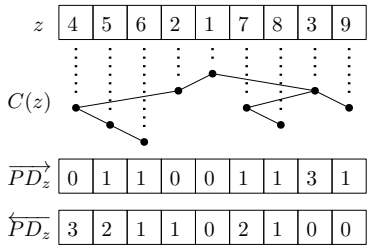


FIG. 7.2 : Une séquence $z = (4, 5, 6, 2, 1, 7, 8, 3, 9)$, son arbre cartésien $C(z)$ et ses tables de distances parentales $\overrightarrow{PD}_z$ et $\overleftarrow{PD}_z$.

2 Recherche approchée

Afin de définir une version approchée du problème CTM, on utilise la notion suivante de transposition sur les séquences :

Définition 7.4 *Swap*

Soient x et y deux séquences de longueur n et $i \in \{1, \ldots, n-1\}$. On note $y = \tau(x, i)$ pour décrire un *swap*, c'est-à-dire :

$$y = \tau(x, i) \text{ si } \begin{cases} x[j] = y[j], \forall j \notin \{i, i+1\} \\ x[i] = y[i+1] \\ x[i+1] = y[i] \end{cases}$$

Ce type de transposition est, par exemple, celle utilisée par l'algorithme de tri à Bulles. Il s'agit d'une opération naturelle sur les permutations et les séquences. On utilise à présent cette notion pour définir la version approchée de CTM.

Définition 7.5 *Recherche approchée : la relation $\overset{\tau}{\approx}_{CT}$*

Soient x et y deux séquences de longueur n, on a $x \overset{\tau}{\approx}_{CT} y$ si :

$$\begin{cases} x \approx_{CT} y, \text{ ou} \\ \\ \exists\, x',\ y', \exists\, i \in \{1, \ldots, n-1\}, \begin{cases} x' \approx_{CT} x, \\ y' \approx_{CT} y, \\ x' = \tau(y', i) \\ \text{et } y' = \tau(x', i) \end{cases} \end{cases}$$

La FIG. FIG.7.3 permet de comprendre l'effet d'un swap sur les arbres cartésiens et montre un exemple de séquences en relation selon $\overset{\tau}{\approx}_{CT}$.

2.1 Caractérisation d'une transposition sur les tables des distances parentales

Dans cette section, on caractérise l'effet du swap appliqué à la position i dans une séquence x de longueur n sur les tables $\overrightarrow{PD}_x$ et $\overleftarrow{PD}_x$. On obtient ainsi une idée précise de ce à quoi les tables $\overrightarrow{PD}_y$ et $\overleftarrow{PD}_y$ d'une séquence y peuvent ressembler, lorsque $y \overset{\tau}{\approx}_{CT} x$. La FIG. 7.4 résume l'ensemble de zones que l'on va caractériser. Cette figure sert de définition aux variables $\overrightarrow{a_x}$, $\overrightarrow{b_x}$, ... Nous utiliserons ensuite cette caractérisation pour obtenir un algorithme qui résout le problème de la recherche approchée.

Le premier Lemme décrit comment les tables sont modifiées aux positions i et $i+1$. L'information importante à retenir de ce résultat est que les valeurs des tables de y sont totalement déterminées par les valeurs des tables de x, à l'exception de $\overrightarrow{a_y}$ (resp. $\overleftarrow{a_y}$), qui peut prendre plusieurs valeurs possibles.

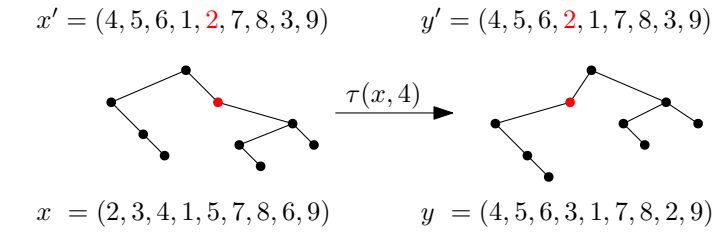


FIG. 7.3 : On a $x \overset{\tau}{\approx}_{CT} y$. Le swap est effectué entre les séquences x' (qui partage le même arbre cartésien que x) et y' (qui partage le même arbre cartésien que y). Un swap à la position 4 déplace le noeud rouge du sous-arbre droit de la racine à son sous-arbre gauche. De manière générale, un swap à la position i revient soit à déplacer le descendant le plus à gauche du sous-arbre droit de son parent vers **une** position la plus à droite dans le sous-arbre gauche (si $x[i] < x[i+1]$), soit à faire l'opposé, c'est à dire déplacer le descendant le plus à droite du sous-arbre gauche vers **une** position la plus à gauche dans le sous-arbre droit de son parent. On remarque que les 4 séquences x, x', y, y' sont toutes en relation selon $\overset{\tau}{\approx}_{CT}$

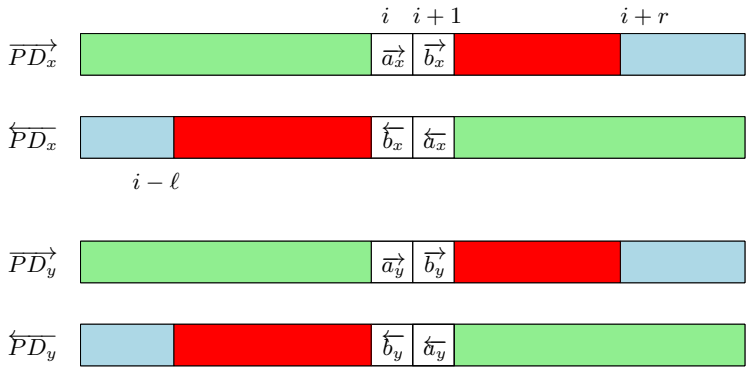


FIG. 7.4 : Cette figure résume les différents lemmes de cette section. Les zones vertes et bleues correspondent par exemple à la Définition 7.7 et au Lemme 7.8. Les valeurs $\overrightarrow{a_x}$, $\overrightarrow{b_x}$, ..., sont les 8 valeurs que l'on trouve dans les tables de x et y aux positions i et $i+1$. Autrement dit $\overrightarrow{PD_x}[i] = \overrightarrow{a_x}$, $\overrightarrow{PD_x}[i+1] = \overrightarrow{b_x}$, ... Les valeurs $i-\ell$ et $i+r$ indiquent respectivement la dernière et la première position de chaque zone bleue.

Lemme 7.6

Les propriétés suivantes sont satisfaites :

Si $x[i] < x[i+1]$

1. $\overleftarrow{b_y} = 1$
2. $\overrightarrow{b_y} = \begin{cases} 0 & \text{si } \overrightarrow{a_x} = 0 \\ \overrightarrow{a_x} + 1 & \text{sinon} \end{cases}$
3. $\overleftarrow{a_y} = \begin{cases} 0 & \text{si } \overleftarrow{b_x} = 0 \\ \overleftarrow{b_x} - 1 & \text{sinon} \end{cases}$
4. $\overrightarrow{a_y} \leq \begin{cases} i-1 & \text{si } \overrightarrow{a_x} = 0 \\ \overrightarrow{a_x} & \text{sinon} \end{cases}$

Si $x[i] > x[i+1]$

1. $\overrightarrow{b_y} = 1$
2. $\overleftarrow{b_y} = \begin{cases} 0 & \text{si } \overleftarrow{a_x} = 0 \\ \overleftarrow{a_x} + 1 & \text{sinon} \end{cases}$
3. $\overrightarrow{a_y} = \begin{cases} 0 & \text{si } \overrightarrow{b_x} = 0 \\ \overrightarrow{b_x} - 1 & \text{sinon} \end{cases}$
4. $\overleftarrow{a_y} \leq \begin{cases} i-1 & \text{si } \overleftarrow{a_x} = 0 \\ \overleftarrow{a_x} & \text{sinon} \end{cases}$

Notons que dans le point 4 du Lemme 7.6, lorsque $\overrightarrow{a_y} \leq \overrightarrow{a_x}$, $\overrightarrow{a_y}$ ne peut pas prendre toutes les valeurs de $\{1, \ldots, \overrightarrow{a_x}\}$, car certaines ne produiraient pas une table des distances parentales valide.

On s'intéresse à présent aux zones de couleurs. Le lecteur ou la lectrice sont invités à utiliser la FIG. 7.4 afin de mieux se forger une intuition des définitions. Afin de les définir d'un bloc, on va d'abord définir les délimiteurs de zones r et l lorsqu'un swap se produit en position i.

$$r = \begin{cases} \overleftarrow{b_x} & \text{si } x[i] < x[i+1] \text{ et } \overleftarrow{b_x} > 1 \\ \overleftarrow{a_x} + 1 & \text{si } x[i] > x[i+1] \text{ et } \overleftarrow{a_x} > 0 \\ n - i + 1 & \text{sinon} \end{cases}$$

$$\ell = \begin{cases} \overrightarrow{a_x} & \text{si } x[i] < x[i+1] \text{ et } \overrightarrow{a_x} > 0 \\ \overrightarrow{b_x} - 1 & \text{si } x[i] > x[i+1] \text{ et } \overrightarrow{b_x} > 1 \\ i & \text{sinon} \end{cases}$$

Ces délimiteurs r et ℓ indiquent de combien de positions il faut se décaler dans la table pour sortir du sous-arbre enraciné en i ou $i+1$, suivant où se situe la plus petite des deux valeurs. En effet, la transposition n'a aucun effet sur les noeuds en dehors de ce sous-arbre (voir FIG. 7.5).

Définition 7.7 *Les zones*

Soit la séquence x et une position de swap i. Pour la table $\overrightarrow{PD}_x$,
- la zone verte est $\overrightarrow{PD}_x[1 \dots i-1]$,
- la zone rouge est $\overrightarrow{PD}_x[i+2 \dots i+r-1]$.
- la zone bleue est $\overrightarrow{PD}_x[i+r \dots n]$,

Pour la table $\overleftarrow{PD}_x$,
- la zone verte est $\overleftarrow{PD}_x[i+2 \dots n]$,
- la zone rouge est $\overleftarrow{PD}_x[i-\ell+1 \dots i-1]$.
- la zone bleue est $\overleftarrow{PD}_x[1 \dots i-\ell]$,

On remarque que les zones bleues peuvent être vides, dans les cas où $\ell = i$ ou $r = n - i + 1$.

Lemme 7.8 *Les zones vertes et bleues*

Les zones vertes $\overrightarrow{PD}_x$ et $\overrightarrow{PD}_y$ (resp. $\overleftarrow{PD}_x$ et $\overleftarrow{PD}_y$) sont égales.
Les zones bleues de $\overrightarrow{PD}_x$ et $\overrightarrow{PD}_y$ (resp. $\overleftarrow{PD}_x$ et $\overleftarrow{PD}_y$) sont égales.

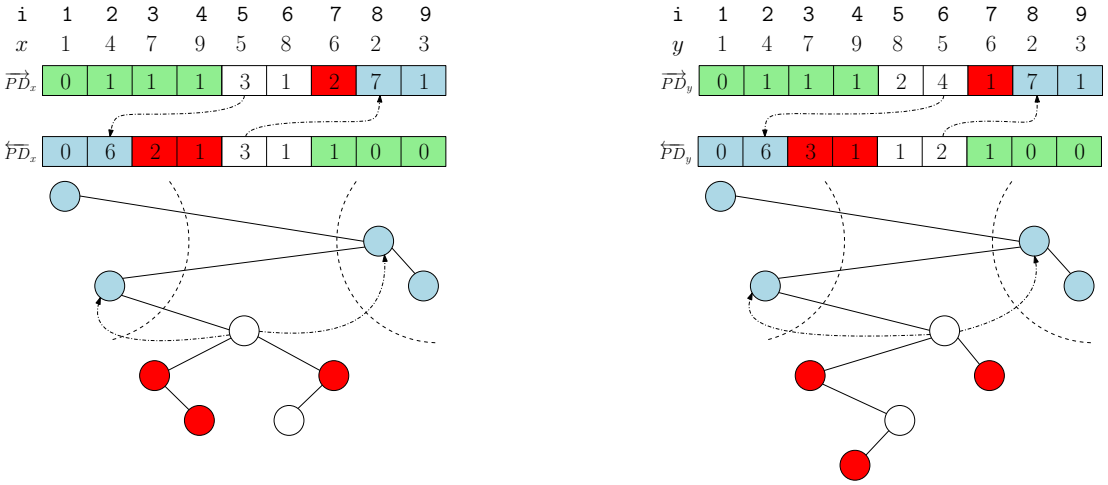


FIG. 7.5 : Sur cette figure, on applique un swap à la position 5 sur x et sur y. Comme on peut le voir sur la figure de gauche, $x[5] < x[6]$, $\ell = \overrightarrow{PD}_x[5]$ et $r = \overleftarrow{PD}_x[5]$, ce qui nous donne les positions $5 + r$ et $5 - \ell$ pour les premières valeurs plus petites que $x[5]$ et, par extension, plus petite que toute valeur entre $5 - \ell$ et $5 + \ell$. Ainsi, toute position plus petite que $5 - \ell$ dans $\overleftarrow{PD}_x$ demeure inchangée par le swap. La même chose s'applique pour toute position plus grande que $5 + r$ dans $\overrightarrow{PD}_x$. Sur la partie droite de la figure, on a $y[5] > y[6]$, $\ell = \overrightarrow{PD}_y[6] - 1$ et $r = \overleftarrow{PD}_y[6] + 1$.

Lemme 7.9 *Les zones rouges*

Pour toute position j dans les zones rouges de $\overrightarrow{PD}_x$ et $\overleftarrow{PD}_x$, on distingue deux cas symétriques :

1. $x[i] < x[i+1]$:
$$\overrightarrow{PD}_y[j] = \begin{cases} \overrightarrow{PD}_x[j] \\ \text{ou} \\ \overrightarrow{PD}_x[j]-1 \end{cases} \quad \text{et } \overleftarrow{PD}_y[j] = \begin{cases} \overleftarrow{PD}_x[j] \\ \text{ou} \\ \overleftarrow{PD}_x[j]+1 \end{cases}$$
2. $x[i] > x[i+1]$:
$$\overrightarrow{PD}_y[j] = \begin{cases} \overrightarrow{PD}_x[j] \\ \text{ou} \\ \overrightarrow{PD}_x[j]+1 \end{cases} \quad \text{et } \overleftarrow{PD}_y[j] = \begin{cases} \overleftarrow{PD}_x[j] \\ \text{ou} \\ \overleftarrow{PD}_x[j]-1 \end{cases} .$$

L'idée de ce lemme est que dans les zones rouges, les tables des distances parentales de deux séquences $x \overset{\tau}{\approx}_{CT} y$ diffèrent d'au plus 1 : le swap peut éloigner ou rapprocher le parent d'une position, soit remplacer le parent par un autre, auquel cas la distance ne change pas. Enfin, nous avons montré que deux swaps à deux positions différentes produisent nécessairement des arbres cartésiens différents.

Lemme 7.10

Soit x une séquence de longueur n et $i, j \in \{1, \dots, n-1\}$, avec $i \neq j$. Alors $\tau(x,i) \not\approx_{CT} \tau(x,j)$.

L'algorithme ci-dessous, est la conséquence des Lemmes 7.6 au 7.10. La fonction candidatSwapPosition à la ligne 11 calcule la position exacte du swap en utilisant les zones vertes (détails dans la section suivante). Il peut y avoir au plus deux positions candidates pour le swap, mais seul l'une d'entre elle peut être valide, d'après le Lemme 7.10. La sous-section suivante décrit comment déterminer ces positions.

Algorithme 11 : $DoubleParentDistanceMethod(p,t)$

Entrées : Deux séquences p et t
Sorties : Le nombre de positions j telles que $t[j \dots j+p-1] \overset{\tau}{\approx}_{CT} p$
$occ \leftarrow 0$;
$(\overrightarrow{PD}_p, \overleftarrow{PD}_p) \leftarrow$ Calculer les tables des distances parentales de p ;
pour tous les $j \in \{1, \dots, |t| - |p| + 1\}$ **faire**
  $(\overrightarrow{PD}_x, \overleftarrow{PD}_x) \leftarrow$ Tables des distances parentales de $x = t[j \dots j+p-1]$;
  si $\overrightarrow{PD}_p = \overrightarrow{PD}_x$ **alors**
    $occ \leftarrow occ + 1$;
  sinon
    pour tous les $i \in candidatSwapPosition(\overrightarrow{PD}_p, \overleftarrow{PD}_p, \overrightarrow{PD}_x, \overleftarrow{PD}_x)$ **faire**
      si *Lemmes 7.6, 7.8 et 7.9 sont vrais pour* p, x *et* i **alors**
        $occ \leftarrow occ + 1$;
retourner occ ;

Théorème 7.11

Soient deux séquences p et t de longueur m et n, l'algorithme 11 a une complexité dans le pire des cas de $\Theta(mn)$ en temps et $\Theta(m)$ en espace.

2.2 Calcul de la position du swap à l'aide des zones vertes

On calcule les positions candidates pour le swap. Si les tables de distances parentales sont égales, alors les deux séquences sont trivialement en relation selon $\overset{\tau}{\approx}_{CT}$. Sinon, l'idée est de "créer un étau" à l'aide des zones vertes. D'après le lemme 7.8, les zones vertes de $\overrightarrow{PD}_x$ et $\overrightarrow{PD}_y$ (resp. $\overleftarrow{PD}_x$ et $\overleftarrow{PD}_y$) sont égales. D'après le lemme 7.6, on en déduit également que $\overrightarrow{b}_x \neq \overrightarrow{b}_y$ et $\overleftarrow{b}_x \neq \overleftarrow{b}_y$. Malheureusement, aucune condition stricte ne pouvant être décrite sur les a, on

recense 4 cas distincts pouvant se produire. en fonction de si $\overrightarrow{a_x}$ est égal à $\overrightarrow{a_y}$ et de si $\overleftarrow{a_x}$ est égal à $\overleftarrow{a_y}$. La FIG. 7.6 présente ces 4 cas possibles.

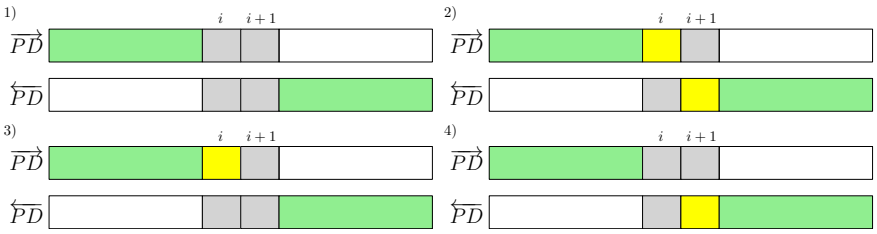


FIG. 7.6 : La table des distances parentales des séquences x et y sont fusionnés en une seule. Si une zone est coloriée soit en vert soit en jaune, alors les tables coïncident. En gris, elle diffèrent. En blanc, le calcul n'est pas utile pour cette partie.

Dans le premier cas, il existe un écart de 2 entre les zones vertes et on peut immédiatement déduire que la première position dans cet écart est la seule position éligible pour un swap potentiel. Dans le second cas, l'écart est de 0, mais on peut une fois de plus situer la position i avec précision.

Dans les cas 3 et 4 il existe un écart de 1. Les deux cas sont alors impossibles à distinguer, ce qui implique qu'il est nécessaire de tester les deux cas : le swap peut avoir eu lieu sur l'une des deux positions. D'après le lemme 7.10, on sait que seule une de ces deux positions peut-être celle un swap. Tout écart supérieur à deux est impossible pour que les deux séquences soient en relation selon $\overset{\tau}{\approx}_{CT}$.

2.3 Le graphe des Swaps

Soit $\mathcal{C}_n$ l'ensemble des arbres cartésiens à n noeuds, qui est égal à l'ensemble des arbres binaires à n noeuds. De plus, $\mathcal{C} = \bigcup_{n\geq 0} \mathcal{C}_n$. Soit $\mathcal{G}_n = (\mathcal{C}_n, E_n)$ le graphe des Swaps des arbres cartésiens, où $\mathcal{C}_n$ est l'ensemble de sommets et E_n l'ensemble des arêtes. Soient x et y deux séquences, on a $\{C(x), C(y)\} \in E_n$ si $x \overset{\tau}{\approx}_{CT} y$. La FIG. 7.7 montre les graphes des Swaps pour $n \leq 4$.

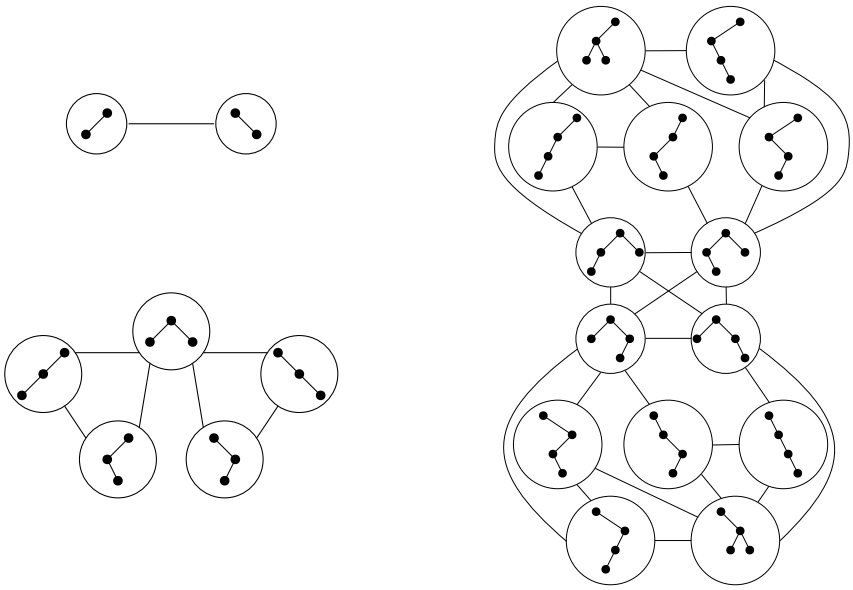


FIG. 7.7 : Graphe des Swaps des arbres cartésiens de taille 2, 3 et 4.

Nous avons étudié l'ensemble des voisins qu'un sommet du graphe peut avoir.

Soit $T \in \mathcal{C}_n$ un arbre cartésien de taille n et $ng(T)$ son ensemble de voisins dans le graphe des Swaps. De plus, pour $i < n$, on note $ng(T, i)$ l'ensemble des arbres obtenus en faisant un swap à la position i sur les séquences associées :

$$ng(T, i) = \{C(y) \in \mathcal{C}_n \mid \exists\ x \text{ tel que } T = C(x) \text{ et } y = \tau(x, i)\}$$

On a également

$$ng(T) = \bigcup_{i=1}^{n-1} ng(T, i)$$

où toutes les unions sont disjointes d'après le Lemme 7.10.

Lemme 7.12

Pour tout arbre T de taille n, on a

$$n - 1 \leq |ng(T)| \leq \lceil 3(n-2) \rceil$$

La borne inférieure est atteinte si la séquence associée à T est strictement croissante ou décroissante.

Le lemme précédent permet d'obtenir une borne inférieure sur le diamètre du graphe des swaps.

Lemme 7.13

Le diamètre du graphe des Swaps $\mathcal{G}_n$ est $\Omega(\frac{n}{\ln n})$.

Le fait que le diamètre du graphe soit suffisament élevé est un argument supplémentaire au fait que cette mesure de distance soit intéressante : les arbres ne sont pas tous "semblables", il existe des arbres très différents les uns des autres. On utilise ensuite la notion de voisinage dans le graphe des Swaps pour définir un nouvel algorithme.

2.4 Un algorithme basé sur l'automate d'Aho-Corasick

La seconde méthode est une adaptation de celle de Park *et al.* [Par+19], qui utilise l'automate d'Aho-Corasick [AC75]. Pour rappel, l'automate d'Aho-Corasick, outil classique en recherche de séquence, est un automate déterministe complet qui reconnaît le langage $\Sigma^* X$, où X est un ensemble fini de séquences finies. Lorsque l'on lit une séquence dans cet automate, passer par un état final implique que la séquence contient une occurence d'un facteur appartenant à l'ensemble X.

Les séquences qui nous intéressent sont celles stockées dans la table des distances parentales. Partant d'une séquence p, on calcule l'ensemble de ses voisins $ng(C(p))$, puis on calcule l'ensemble X des tables des distances parentales associées. Il suffit alors de reprendre la méthode de Park *et al.* pour la reconnaissance de motifs multiples.

Le théorème suivant est obtenu grâce à la borne supérieure du lemme 7.12 et à la Section 4.2 dans [Par+19].

Théorème 7.14

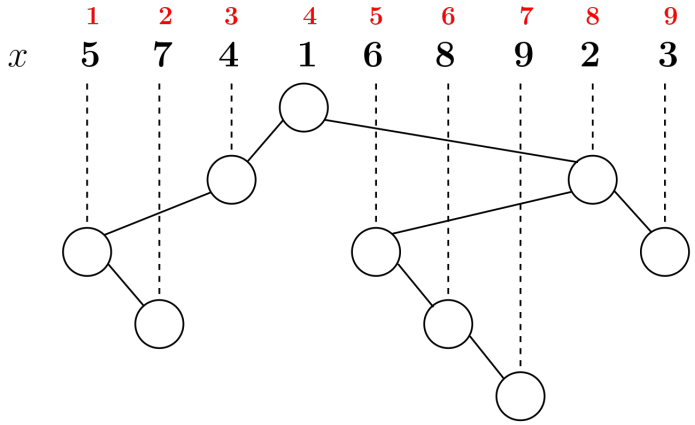
Soient deux séquences p et t de longueur m et n, l'algorithme basé sur l'automate d'Aho-Corasick a une complexité dans le pire des cas de $\mathcal{O}((m^2 + n) \log m)$ en temps et $\mathcal{O}(m^2)$ en espace.

3 La Skipped Number Representation

Avant de présenter la Skipped Number Representation, on présente des fonctions auxiliaires. Soient une séquence x de longueur n et $rmp_x : \{1, \ldots, n\} \mapsto \mathcal{P}(\{1, \ldots, n\})$ la fonction qui associe à chaque position i dans l'arbre cartésien $C(x)$ l'ensemble des positions des noeuds sur la branche droite de son sous-arbre gauche (rmp pour rightmost-path). Notons que, le noeud correspondant à la position 1 n'ayant jamais de sous-arbre gauche, on a toujours $rmp_x(1) = \emptyset$. Symétriquement, on définit $lmp_x : \{1, \ldots, n\} \mapsto \mathcal{P}(\{1, \ldots, n\})$ pour les noeuds de la branche gauche du sous-arbre droit.

Théorème 7.15

Soit une séquence $x[1 \ldots n]$, la fonction rmp_x (resp. lmp_x) peut être entièrement calculée en temps $\mathcal{O}(n)$.



rmq_x	∅	∅	{1,2}	{3}	∅	∅	∅	{5,6,7}	∅
$\overrightarrow{ref_x}$	3	3	4	-1	8	8	8	-1	-1
$\overrightarrow{SN_x}$	0	0	2	1	0	0	0	3	0
lmq_x	{2}	∅	∅	{5,8}	{6}	{7}	∅	{9}	∅
$\overleftarrow{ref_x}$	-1	1	-1	-1	4	5	6	4	8
$\overleftarrow{SN_x}$	1	0	0	2	1	1	0	1	0

FIG. 7.8 : L'arbre cartésien $x = 5, 7, 4, 1, 6, 8, 9, 2, 3$ et ses tables associées.

Définition 7.16

Soit une séquence x de longueur n, soit $\overrightarrow{ref_x} : \{1, \ldots, n\} \mapsto \{2, \ldots, n\} \cup \{-1\}$ la fonction qui associe à chaque position i dans l'arbre cartésien $C(x)$ la position j de la racine du sous-arbre C', tel que i est sur la branche droite du sous-arbre gauche de C'. Formellement, on a

$$\overrightarrow{ref_x}(i) = \begin{cases} j, \text{ si } i \in rmp_x(j) \\ -1, \text{ sinon.} \end{cases} \tag{7.1}$$

Lorsque $\overrightarrow{ref_x}(i) = j$, j est appelé le *référent* de i. Notons que pour toute séquence $x[1 \ldots n]$ et pour $1 \leq i \leq n$, on a $i < \overrightarrow{ref_x}(i)$ lorsque $\overrightarrow{ref_x}(i) \neq -1$.

Théorème 7.17

Soit une séquence $x[1 \ldots n]$, la fonction $\overrightarrow{ref_x}$ peut être entièrement calculée en temps $\mathcal{O}(n)$.

On définit à présent une nouvelle représentation linéaire d'un arbre cartésien appelée *Skipped-number* representation.

Définition 7.18 Skipped-number *representation*

Soit une séquence $x[1 \ldots n]$, une *Skipped-number* representation de x est une des séquences $\overrightarrow{SN}_x[1 \ldots n]$ et $\overleftarrow{SN}_x[1 \ldots n]$, définies comme suit :

$$\overrightarrow{SN}_x[i] = |rmp_x(i)| \text{ et } \overleftarrow{SN}_x[i] = |lmp_x(i)|$$

Notons que $\forall i, j$, on a $rmp_x(i) \cap rmp_x(j) = \emptyset$. Ainsi, lorsque l'on applique une transformation sur une séquence x, $\overrightarrow{SN}_x[i]$ est modifiée si et seulement si $rmp_x(i)$ est modifiée.

Une notion similaire de la *Skipped-number* representation apparaît dans le livre d'Ohlebusch [Ohl13], au chapitre 3.

Théorème 7.19

Il existe une bijection entre la *Skipped-number* representation et la structure d'arbre cartésien. La *Skipped-number* representation d'une séquence $x[1 \dots n]$ peut- être calculée en temps $\mathcal{O}(n)$.

3.1 Caractérisation d'une transposition sur la *Skipped-number* representation

La *Skipped-number* representation a l'avantage suivant sur la table des distances parentales : il est possible de majorer le nombre de modifications effectuées sur la table lors d'un swap sur une séquence.

Lemme 7.20

Soient deux séquences x et y de longueur n, tel qu'il existe $i \in \{1, \dots, n-1\}$ et $y = \tau(x, i)$. Il existe au plus 3 positions j telles que $\overrightarrow{SN}_x[j] \neq \overrightarrow{SN}_y[j]$, avec $j \in \{i, i+1, \overrightarrow{ref}_x(i), \overrightarrow{ref}_x(i+1), \overrightarrow{ref}_y(i), \overrightarrow{ref}_y(i+1)\}$.

Pour être précis, il est possible de montrer que, lors d'un swap en position i :

- il y a nécessairement une modification de $\overrightarrow{SN}_x[i+1]$.
- il peut y avoir une modification de $\overrightarrow{SN}_x[i]$.
- il peut y avoir une modification de $\overrightarrow{SN}_x[j]$, $j \in \{\overrightarrow{ref}_x(i), \overrightarrow{ref}_x(i+1), \overrightarrow{ref}_y(i), \overrightarrow{ref}_y(i+1)\}$.

Le nombre total de modifications sur $\overrightarrow{SN}_x$ lors d'un swap, peut-être égal à 1, 2 ou 3. La Fig. 7.9 illustre les différents cas possibles.

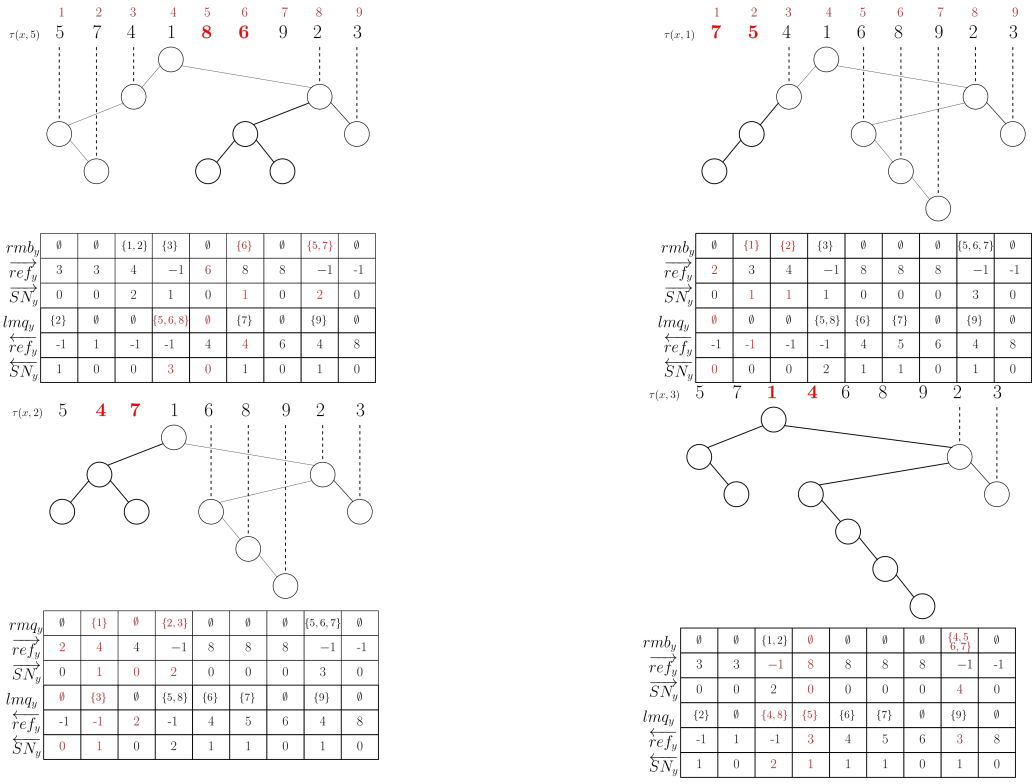


Fig. 7.9 : On montre l'effet d'un swap sur la séquence x de la Fig. 7.8 aux positions 5, 1, 2 et 3. Les valeurs qui diffèrent de celles des tables de la Fig. 7.8 sont notées en rouge.

Il est possible de prédire précisément les valeurs modifiées dans $\overrightarrow{SN}_x$ et ce, en temps constant, si la fonction $\overrightarrow{ref}_x$ a été précalculée.

Lemme 7.21

Soient deux séquences x et y. Pour tout $i \in \{1, \ldots, n-1\}$, on a $\overrightarrow{SN}_y = \overrightarrow{SN}_{\tau(x,i)}$ si et seulement si la valeur $\overrightarrow{SN}_y[j]$ pour

$$j \in \{i, i+1, \overrightarrow{ref_x}(i), \overrightarrow{ref_x}(i+1), \overrightarrow{ref_x}(i+1)\}$$

est déterminée par les valeurs

$$\{k, \overrightarrow{SN}_x[i], \overrightarrow{SN}_x[i+1], \overrightarrow{SN}_x[\overrightarrow{ref_x}(i)], \overrightarrow{SN}_x[\overrightarrow{ref_x}(i+1)], \overrightarrow{SN}_x[\overrightarrow{ref_y}(i+1)]\}$$

4 Perspectives

Plusieurs pistes sont en cours d'études pour produire un algorithme efficace en utilisant la *Skipped-number* representation. Dans un travail commun avec Gadi Landau, Samah Idrees Ghazawi, Bastien Auvray, Richard Groult et enfin Thierry Lecroq, plusieurs pistes ont été évoquées : réadapter les algorithmes de Aho-Corasick et de Boyer-Moore, ou encore développer une méthode `ad-hoc`. Bastien Auvray commencera son doctorat à la rentrée prochaine, sous la direction de Thierry Lecroq et co-encadré par Richard Groult et moi-même.

L'étude en moyenne de l'algorithme 11 devrait aussi permettre de démontrer que l'algorithme a un comportement plus efficace en pratique que ce que l'étude du pire des cas laisse supposer.

Sungmin Kim et Yo-Sub Han, dans un article intitulé *Approximate Cartesian Tree Pattern Matching*, ont définis des algorithmes de recherche approchés, où la distance d'édition entre deux motifs est bornée. Il serait intéressant de comparer leur notion de distance à la notre, en considérant l'équivalent du Swap Graph, en reliant deux arbres cartésiens lorsque leur distance d'édition est de 1.

8

Conclusion et Perspectives

Ce manuscrit montre une partie des travaux de recherches effectués depuis l'obtention de mon poste de maître de conférences. "Comment réussir à réaliser une analyse théorique des algorithmes qui permette d'expliquer le comportement de ces derniers en pratique ?" est une question que je me pose depuis mon doctorat.

La réponse apportée dans ce manuscrit est le fruit d'essais effectués au cours des années, dont certains se sont révélés être des impasses.

- J'ai par exemple tenté d'utiliser des méthodes apprentissage supervisé afin de détecter le meilleur algorithme à utiliser en fonction d'une entrée donnée. Le principe était le suivant : on vectorise les objets combinatoires passés en entrée de l'algorithme et on apprend au réseau de neurones à prédire quel est l'algorithme qui effectue le temps de calcul minimum (le réseau renvoie un vecteur où un score est attribué à chaque algorithme). Le résultat le plus efficace obtenu a consisté à transformer des permutations en images en deux dimension et à utiliser les réseaux de classification d'images implantés dans *pyTorch*. Le meilleur classifieur permettait de prédire le meilleur algorithme à utiliser avec une fiabilité supérieure à 90%. Toutefois, l'apport théorique était nul car l'utilisation des classifieurs ne permettait pas d'apporter une explication supplémentaire. De plus, en pratique, l'utilisation du classifieur ne permettait pas de gagner en temps de calcul, ce coût étant lui même prohibitif au vu de l'efficacité des algorithmes de tri. J'ai donc délaissé totalement cette voie après avoir passé plusieurs mois dessus.
- Je me suis également tourné sur l'influence de la présence ou l'absence d'un motif dans un objet combinatoire sur les algorithmes qui s'y appliquent. A ce titre, j'ai développé avec Gwendal Collet et Alice Jacquot un algorithme qui produit la spécification combinatoire, aisément convertible en système algébrique, d'arbres dont les nombres n de noeuds et k d'occurrences d'un motif donné en entrée sont fixés. En utilisant la méthode récursive, il est possible d'obtenir un générateur aléatoire uniforme pour n et k fixé. L'article en question ne fut accepté qu'en tant que poster à AOFA 2014.
- Dans la recherche de méthodes efficaces de génération aléatoire, j'ai travaillé avec Olivier Bodini et Camille Cotti sur la parallélisation d'algorithmes de génération d'arbres planaires à l'aide de processus de Galton-Watson. Il s'agissait avant tout d'un travail d'ingénierie dans lequel les bits aléatoires étaient tirés par un processus dédié, qui fournissait des pages de bits aléatoires à la demande aux processus chargés de construire l'arbre. Les résultats, trop expérimentaux, n'ont pas intéressé les conférences et journaux d'informatique théorique. Les conférences dédiées aux techniques de parallélisation n'ont pas été intéréssées par cette problématique.

Je suis également persuadé qu'il est possible d'utiliser des techniques de fouilles de données pour faciliter le travail des chercheuses et chercheurs en analyse d'algorithmes, en particulier l'utilisation d'algorithmes d'énumération de motifs émergeants. Un motif émergeant est un motif qui apparaît (fréquemment) dans un ensemble d'objets et qui est absent (ou peu fréquent) dans le complémentaire de cet ensemble. Imaginons cette notion appliquée à l'analyse d'algorithmes. On engendre aléatoirement des objets combinatoires, on leur applique l'algorithme que l'on souhaite étudier, puis on partitionne l'ensemble des objets engendrés en fonction de l'efficacité de l'algorithme. On applique ensuite un algorithme d'extraction de motifs émergeants afin de détecter quels sont les motifs qui semblent avoir une influence sur le comportement de l'algorithme. Par exemple, dans [FN16], les auteurs montrent que la complexité moyenne de l'algorithme de Brzozowski de minimisation d'automates dépend du nombre de cycle disjoints de grandes tailles, accessibles depuis un état initial. Un tel constat a du prendre du temps. Une méthode automatique aurait permis de trouver plus rapidement la propriété intéressante à étudier pour obtenir leur résultat. J'ai déjà abordé cette question avec Loick Lhote et François Rioult et espère travailler avec eux dans un avenir proche.

La méthode développée dans ce manuscrit et exposée dans l'interlude en est encore à ses prémisses. La notion de fonction *déterminante* semble triviale une fois exposée (sans pourtant avoir été exploitée auparavant) et celle de fonction *intéressante* est encore trop floue. Il conviendrait de mieux caractériser les types de fonctions/propriétés pour lesquelles

$$\int_{\pi\in D} \mathbb{P}(\pi) CoutMoyen(\pi)^2 d\pi - \left(\int_{\pi\in D} \mathbb{P}(\pi) CoutMoyen(\pi) d\pi\right)^2$$

c'est-à-dire telles que la variance du coût moyen reste "suffisamment faible" pour que l'on puisse considérer que les fonctions/propriétés aient un sens. On pourrait également définir des fonctions F *quasi-déterminante* telles que, pour un ε fixé, $\forall t$, $\forall n$, $\forall \pi_1, \pi_2 \in D_{t,n}$, on ait

$$CoutMoyen(\pi_1) \leq CoutMoyen(\pi_1) + \varepsilon$$

Il serait alors possible une borne supérieure de la valeur de l'intégrale en déterminant le coût moyen de l'algorithme pour une seule distribution.

Mon objectif est de trouver des collaboratrices et collaborateurs, notamment en statistiques, pour explorer cette partie. Le générateur aléatoire développé dans le chapitre 3 peut être utilisé dans des contextes plus variés que ceux exposés dans ce manuscrit et il conviendrait d'explorer plus profondément les possibilités. Mon objectif à moyen et long terme est donc de développer cette méthodologie, de l'enrichir et de parvenir à décrire des familles d'algorithmes dont le comportement moyen est régi par un même paramètre déterminant.

Bibliography

[AM02] Dimitris Achlioptas et Cristopher Moore. « On the 2-colorability of random hypergraphs ». In : *Randomization and Approximation Techniques in Computer Science : 6th International Workshop, RANDOM 2002 Cambridge, MA, USA, September 13–15, 2002 Proceedings 5*. Springer. 2002, p. 78-90 (cf. p. 44).

[AŽ95] Dragan M Acketa et Joviša D Žunić. « On the maximal number of edges of convex digital polygons included into an m× m-grid ». In : *Journal of Combinatorial Theory, Series A* 69.2 (1995), p. 358-368 (cf. p. 29, 34).

[AIS93] Rakesh Agrawal, Tomasz Imieliński et Arun Swami. « Mining association rules between sets of items in large databases ». In : *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 1993, p. 207-216 (cf. p. 43).

[AS+94] Rakesh Agrawal, Ramakrishnan Srikant et al. « Fast algorithms for mining association rules ». In : *Proc. 20th int. conf. very large data bases, VLDB*. T. 1215. Santiago. 1994, p. 487-499 (cf. p. 47).

[AC75] A. V. Aho et M. J. Corasick. « Efficient string matching : an aid to bibliographic search ». In : *Commun. ACM* 18.6 (1975), p. 333-340 (cf. p. 70, 77).

[ARS97a] Laurent Alonso, Jean-Luc Remy et René Schott. « A linear-time algorithm for the generation of trees ». In : *Algorithmica* 17 (1997), p. 162-182 (cf. p. 8).

[ARS97b] Laurent Alonso, Jean-Luc Remy et René Schott. « Uniform generation of a Schröder tree ». In : *Information processing letters* 64.6 (1997), p. 305-308 (cf. p. 8).

[Ami+97] Amihood Amir, Yonatan Aumann, Gad M. Landau, Moshe Lewenstein et Noa Lewenstein. « Pattern Matching with Swaps ». In : *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*. IEEE Computer Society, 1997, p. 144-153 (cf. p. 71).

[AG86] Alberto Apostolico et Raffaele Giancarlo. « The Boyer–Moore–Galil String Searching Strategies Revisited ». In : *SIAM Journal on Computing* 15.1 (1986), p. 98-105 (cf. p. 66).

[ANP16] Nicolas Auger, Cyril Nicaud et Carine Pivoteau. « Good Predictions Are Worth a Few Comparisons ». In : *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*. Sous la dir. de Nicolas Ollinger et Heribert Vollmer. T. 47. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 12 :1-12 :14 (cf. p. 3, 4, 64).

[BBJ13] Axel Bacher, Olivier Bodini et Alice Jacquot. « Exact-size sampling for motzkin trees in linear time via boltzmann samplers and holonomic specification ». In : *2013 Proceedings of the Tenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM. 2013, p. 52-61 (cf. p. 8).

[BMR03] James Bailey, Thomas Manoukian et Kotagiri Ramamohanarao. « A Fast Algorithm for Computing Hypergraph Transversals and its Application in Mining Emerging Patterns ». In : *Proceedings of the Third IEEE International Conference on Data Mining*. ICDM '03. Washington, DC, USA : IEEE Computer Society, 2003, p. 485-489. isbn : 0-7695-1978-4 (cf. p. 48).

[BZ06] James Bailey et Zhou Zhu. « Fast discovery of interesting collections of web services ». In : *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*. IEEE. 2006, p. 152-160 (cf. p. 43).

[BP92] Imre Bárány et János Pach. « On the number of convex lattice polygons ». In : *Combinatorics, Probability and Computing* 1.4 (1992), p. 295-302 (cf. p. 28).

[BN07] Frédérique Bassino et Cyril Nicaud. « Enumeration and random generation of accessible automata ». In : *Theoretical Computer Science* 381.1-3 (2007), p. 86-104 (cf. p. 5).

[Ber89] C. Berge. *Hypergraphs*. T. 45. North-Holland, Amsterdam : North-Holland Mathematical Library, 1989 (cf. p. 3, 42, 45-47).

[BS18] Mónica Blanco et Francisco Santos. « Enumeration of lattice 3-polytopes by their number of lattice points ». In : *Discrete & Computational Geometry* 60.3 (2018), p. 756-800 (cf. p. 28).

[BS19] Mónica Blanco et Francisco Santos. « Non-spanning lattice 3-polytopes ». In : *Journal of Combinatorial Theory, Series A* 161 (2019), p. 112-133 (cf. p. 28).

[Bod+13a] Olivier Bodini, Ph Duchon, Alice Jacquot et L Mutafchiev. « Asymptotic analysis and random sampling of digitally convex polyominoes ». In : *Discrete Geometry for Computer Imagery : 17th IAPR International Conference, DGCI 2013, Seville, Spain, March 20-22, 2013. Proceedings 17*. Springer. 2013, p. 95-106 (cf. p. 5).

[Bod+13b] Olivier Bodini, Ph Duchon, Alice Jacquot et L Mutafchiev. « Asymptotic analysis and random sampling of digitally convex polyominoes ». In : *Discrete Geometry for Computer Imagery : 17th IAPR International Conference, DGCI 2013, Seville, Spain, March 20-22, 2013. Proceedings 17*. Springer. 2013, p. 95-106 (cf. p. 28).

[BFP10] Olivier Bodini, Éric Fusy et Carine Pivoteau. « Random sampling of plane partitions ». In : *Combinatorics, Probability and Computing* 19.2 (2010), p. 201-226 (cf. p. 5).

[BGJ13] Olivier Bodini, Danièle Gardy et Alice Jacquot. « Asymptotics and random sampling for BCI and BCK lambda terms ». In : *Theoretical Computer Science* 502 (2013), p. 227-238 (cf. p. 5).

[BP10] Olivier Bodini et Yann Ponty. « Multi-dimensional Boltzmann sampling of languages ». In : *Discrete Mathematics & Theoretical Computer Science* Proceedings (2010) (cf. p. 8).

[Bog+15] Tristram Bogart, Christian Haase, Milena Hering, Benjamin Lorenz, Benjamin Nill, Andreas Paffenholz, Günter Rote, Francisco Santos et Hal Schenck. « Finitely many smooth d-polytopes with n lattice points ». In : *Israel journal of mathematics* 207 (2015), p. 301-329 (cf. p. 28).

[BEM08] Endre Boros, Khaled Elbassioni et Kazuhisa Makino. « On Berge Multiplication for Monotone Boolean Dualization ». In : juill. 2008. isbn : 978-3-540-70574-1. doi : 10.1007/978-3-540-70575-8_5 (cf. p. 47).

[BG03] Endre Boros et Vladimir Gurvich. « On Nash-solvability in pure stationary strategies of finite games with perfect information which may have cycles ». In : *Mathematical social sciences* 46.2 (2003), p. 207-241 (cf. p. 43).

[BLS99] Andreas Brandstädt, Van Bang Le et Jeremy P Spinrad. *Graph classes : a survey*. SIAM, 1999 (cf. p. 43).

[BV97] Michel Brion et Michele Vergne. « Lattice points in simple polytopes ». In : *Journal of the American Mathematical Society* 10.2 (1997), p. 371-392 (cf. p. 28).

[BM14] Nicolas Broutin et Jean-François Marckert. « Asymptotics of trees with a prescribed degree sequence and applications ». In : *Random Structures & Algorithms* 44.3 (2014), p. 290-316 (cf. p. 12).

[BGM16] Winfried Bruns, Joseph Gubeladze et Mateusz Michałek. « Quantum jumps of normal polytopes ». In : *Discrete & Computational Geometry* 56.1 (2016), p. 181-215 (cf. p. 28).

[Cha+04] Jean-Marc Champarnaud, Georges Hansel, Thomas Paranthoën et Djelloul Ziadi. « Random Generation Models for NFAs ». In : *J. Autom. Lang. Comb.* 9.2/3 (2004), p. 203-216. doi : 10.25596/jalc-2004-203. url : https://doi.org/10.25596/jalc-2004-203 (cf. p. 3).

[Cho+15] Sukhyeun Cho, Joong Chae Na, Kunsoo Park et Jeong Seop Sim. « A fast algorithm for order-preserving pattern matching ». In : *Information Processing Letters* 115.2 (2015), p. 397-402 (cf. p. 70).

[CR20] Maxime Crochemore et Luís M.S. Russo. « Cartesian and Lyndon trees ». In : *Theoret. Comput. Sci.* 806 (2020), p. 1-9. issn : 0304-3975 (cf. p. 70).

[Dam06] Peter Damaschke. « Parameterized enumeration, transversals, and imperfect phylogeny reconstruction ». In : *Theoretical Computer Science* 351.3 (2006), p. 337-350 (cf. p. 43).

[DM16] Alberto Del Pia et Carla Michini. « On the diameter of lattice polytopes ». In : *Discrete & Computational Geometry* 55 (2016), p. 681-687 (cf. p. 29).

[DLW14] Erik D. Demaine, Gad M. Landau et Oren Weimann. « On Cartesian Trees and Range Minimum Queries ». In : *Algorithmica* 68.3 (2014), p. 610-625 (cf. p. 70).

[DPT10] Alain Denise, Yann Ponty et Michel Termier. « Controlled non-uniform random generation of decomposable structures ». In : *Theoretical Computer Science* 411.40-42 (2010), p. 3527-3552 (cf. p. 8).

[DDT14] Olivier Devillers, Philippe Duchon et Rémy Thomasse. « A generator of random convex polygons in a disc ». In : *AofA 2014-25th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms*. 2014 (cf. p. 28).

[Dev86] Luc Devroye. « Sample-based non-uniform random variate generation ». In : *Proceedings of the 18th conference on Winter simulation*. 1986, p. 260-265 (cf. p. 8).

[DMO18] Antoine Deza, George Manoussakis et Shmuel Onn. « Primitive zonotopes ». In : *Discrete & Computational Geometry* 60 (2018), p. 27-39 (cf. p. 29, 34).

[DP18] Antoine Deza et Lionel Pournin. « Improved bounds on the diameter of lattice polytopes ». In : *Acta Mathematica Hungarica* 154 (2018), p. 457-469 (cf. p. 29).

[DDP09] Alicia Dickenstein, Sandra Di Rocco et Ragni Piene. « Classifying smooth lattice polytopes via toric fibrations ». In : *Advances in Mathematics* 222.1 (2009), p. 240-254 (cf. p. 28).

[Did19] Gilles Didier. « Optimal pattern matching algorithms ». In : *Journal of Complexity* 51 (2019), p. 79-109. issn : 0885-064X (cf. p. 60).

[DT17] Gilles Didier et Laurent Tichit. « Designing optimal- and fast-on-average pattern matching algorithms ». In : *Journal of Discrete Algorithms* 42 (2017), p. 45-60. issn : 1570-8667 (cf. p. 60).

[Don11] traduit par Patrick Cégielski Donald E. Knuth. *Algorithmes*. CSLI Publications, 2011 (cf. p. 2).

[DL05] Guozhu Dong et Jinyan Li. « Mining border descriptions of emerging patterns from dataset pairs ». In : *Knowledge and Information Systems* 8 (2 2005), p. 178-202. issn : 0219-1377 (cf. p. 48).

[Duc+04] Philippe Duchon, Philippe Flajolet, Guy Louchard et Gilles Schaeffer. « Boltzmann samplers for the random generation of combinatorial structures ». In : *Combinatorics, Probability and Computing* 13.4-5 (2004), p. 577-625 (cf. p. 4, 8).

[DF10] Andrzej Dudek et Alan Frieze. *Loose Hamilton Cycles in Random k-Uniform Hypergraphs*. 2010 (cf. p. 44).

[EG02] Thomas Eiter et Georg Gottlob. « Hypergraph Transversal Computation and Related Problems in Logic and AI ». In : *JELIA*. Sous la dir. de Sergio Flesca, Sergio Greco, Nicola Leone et Giovambattista Ianni. T. 2424. Lecture Notes in Computer Science. Springer, 2002, p. 549-564. isbn : 3-540-44190-5 (cf. p. 43).

[ER59] P. Erdös et A. Rényi. « On random graphs. I ». In : *Publ. Math. Debrecen* 6 (1959), p. 290-297 (cf. p. 43).

[FL10] Simone Faro et Thierry Lecroq. « The Exact String Matching Problem : a Comprehensive Experimental Evaluation ». In : *CoRR* (2010) (cf. p. 60).

[Far+16] Simone Faro, Thierry Lecroq, Stefano Borzì, Simone Di Mauro et Alessandro Maggio. « The String Matching Algorithms Research Tool ». In : *Proceedings of the Prague Stringology Conference 2016*. Sous la dir. de Jan Holub et Jan Žďárek. Czech Technical University in Prague, Czech Republic, 2016, p. 99-111. isbn : 978-80-01-05996-8 (cf. p. 65).

[Far+23] Simone Faro, Thierry Lecroq, Kunsoo Park et Stefano Scafiti. « On the Longest Common Cartesian Substring Problem ». In : *Comput. J.* 66.4 (2023), p. 907-923 (cf. p. 70).

[FP18] Simone Faro et Arianna Pavone. « An Efficient Skip-Search Approach to Swap Matching ». In : *Comput. J.* 61.9 (2018), p. 1351-1360 (cf. p. 71).

[FN16] Sven De Felice et Cyril Nicaud. « Average Case Analysis of Brzozowski's Algorithm ». In : *Int. J. Found. Comput. Sci.* 27.2 (2016), p. 109-126. doi : 10.1142/S0129054116400025. url : https://doi.org/10.1142/S0129054116400025 (cf. p. 4, 82).

[FS95] Philippe Flajolet et Robert Sedgewick. « Mellin transforms and asymptotics : Finite differences and Rice's integrals ». In : *Theoretical Computer Science* 144.1-2 (1995), p. 101-124 (cf. p. 54).

[FS09] Philippe Flajolet et Robert Sedgewick. *Analytic combinatorics*. cambridge University press, 2009 (cf. p. 4).

[FZV94] Philippe Flajolet, Paul Zimmermann et Bernard Van Cutsem. « A calculus for the random generation of labelled combinatorial structures ». In : *Theoretical Computer Science* 132.1-2 (1994), p. 1-35 (cf. p. 4, 8).

[FJS07] Frantisek Franek, Christopher G. Jennings et W.F. Smyth. « A simple fast hybrid pattern-matching algorithm ». In : *Journal of Discrete Algorithms* 5.4 (2007). Selected papers from Combinatorial Pattern Matching 2005, p. 682-695. issn : 1570-8667 (cf. p. 65).

[FK96] Michael L. Fredman et Leonid Khachiyan. « On the Complexity of Dualization of Monotone Disjunctive Normal Forms ». In : *J. Algorithms* 21.3 (1996), p. 618-628 (cf. p. 3, 43, 48, 57).

[Fus09] Éric Fusy. « Uniform random sampling of planar graphs in linear time ». In : *Random Structures & Algorithms* 35.4 (2009), p. 464-522 (cf. p. 5).

[GB85] Hector Garcia-Molina et Daniel Barbara. « How to assign votes in a distributed system ». In : *Journal of the ACM (JACM)* 32.4 (1985), p. 841-860 (cf. p. 43).

[Gar05] Daniele Gardy. « Random boolean expressions ». In : *Discrete Mathematics & Theoretical Computer Science* Proceedings (2005) (cf. p. 5).

[Gun+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen et Ram Sewak Sharma. « Discovering all most specific sentences ». In : *ACM Transactions on Database Systems (TODS)* 28.2 (2003), p. 140-174 (cf. p. 43).

[Gun+97] Dimitrios Gunopulos, Heikki Mannila, Roni Khardon et Hannu Toivonen. « Data mining, hypergraph transversals, and machine learning ». In : *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 1997, p. 209-216 (cf. p. 43).

[GS93] CG Gunther et WR Scheider. « Entropy as a function of alphabet size ». In : *Proceedings. IEEE International Symposium on Information Theory*. IEEE. 1993, p. 70-70 (cf. p. 18).

[Hag08] Matthias Hagen. « Algorithmic and Computational Complexity Issues of MONET ». Dissertation. Institut für Informatik, Friedrich-Schiller-Universität Jena, déc. 2008 (cf. p. 45, 48).

[HKS08] Utz-Uwe Haus, Steffen Klamt et Tamon Stephen. « Computing knock-out strategies in metabolic networks ». In : *Journal of Computational Biology* 15.3 (2008), p. 259-268 (cf. p. 43).

[HBC07] Céline Hébert, Alain Bretto et Bruno Crémilleux. « A Data Mining Formalization to Improve Hypergraph Minimal Transversal Computation ». In : *Fundam. Inf.* 80 (4 2007), p. 415-433. ISSN : 0169-2968 (cf. p. 46, 47).

[Hoa61] C. A. R. Hoare. « Algorithm 64 : Quicksort ». In : *Commun. ACM* 4.7 (1961), p. 321. ISSN : 0001-0782. DOI : 10.1145/366622.366644. URL : https://doi.org/10.1145/366622.366644 (cf. p. 3).

[Kan+14] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary et Lhouari Nourine. « On the enumeration of minimal dominating sets and related notions ». In : *SIAM Journal on Discrete Mathematics* 28.4 (2014), p. 1916-1929 (cf. p. 43).

[KLM07] Naouel Karam, Serge Linckels et Christoph Meinel. « Semantic composition of lecture subparts for a personalized e-learning ». In : *The Semantic Web : Research and Applications : 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007. Proceedings 4*. Springer. 2007, p. 716-728 (cf. p. 43).

[KS05] Dimitris J. Kavvadias et Elias C. Stavropoulos. « An efficient algorithm for the transversal hypergraph generation. » English. In : *J. Graph Algorithms Appl.* 9.2 (2005), p. 239-264 (cf. p. 48).

[KP04] Dogan Kesdogan et Lexi Pimenidis. « The hitting set attack on anonymity protocols ». In : *International Workshop on Information Hiding*. Springer. 2004, p. 326-339 (cf. p. 43).

[Kha01] Leonid Khachiyan. « Transversal hypergraphs and families of polyhedral cones ». In : *Advances in Convex Analysis and Global Optimization : Honoring the Memory of C. Caratheodory (1873–1950)* (2001), p. 105-118 (cf. p. 43).

[Kha+08] Leonid Khachiyan, Endre Boros, Khaled Elbassioni et Vladimir Gurvich. « Generating all minimal integral solutions to AND–OR systems of monotone inequalities : Conjunctions are simpler than disjunctions ». In : *Discrete applied mathematics* 156.11 (2008), p. 2020-2034 (cf. p. 43).

[Kim+14] Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi et Takeshi Tokuyama. « Order-preserving matching ». In : *Theoretical Computer Science* 525 (2014). Advances in Stringology, p. 68-79. ISSN : 0304-3975. DOI : https://doi.org/10.1016/j.tcs.2013.10.006. URL : https://www.sciencedirect.com/science/article/pii/S0304397513007585 (cf. p. 70).

[KC21] Sung-Hwan Kim et Hwan-Gue Cho. « A Compact Index for Cartesian Tree Matching ». In : *CPM*. Wrocław, Poland, 2021, 18 :1-18 :19 (cf. p. 70).

[KSG07] Steffen Klamt, Julio Saez-Rodriguez et Ernst D Gilles. « Structural and functional analysis of cellular networks with CellNetAnalyzer ». In : *BMC systems biology* 1.1 (2007), p. 1-13 (cf. p. 43).

[KO92] Peter Kleinschmidt et Shmuel Onn. « On the diameter of convex polytopes. » In : *Discret. Math.* 102.1 (1992), p. 75-77 (cf. p. 29).

[KY76] Donald Knuth et Andrew C. Yao. « The complexity of nonuniform random number generation ». In : *Algorithm and Complexity, New Directions and Results* (1976) (cf. p. 8).

[KMP77] Donald E. Knuth, James H. Morris Jr. et Vaughan R. Pratt. « Fast Pattern Matching in Strings ». In : *SIAM Journal on Computing* 6.2 (1977), p. 323-350 (cf. p. 3, 66, 67, 70, 71).

[KNR19] Florent Koechlin, Cyril Nicaud et Pablo Rotondo. « Uniform Random Expressions Lack Expressivity ». In : *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*. Sous la dir. de Peter Rossmanith, Pinar Heggernes et Joost-Pieter Katoen. T. 138. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 51 :1-51 :14. DOI : 10.4230/LIPIcs.MFCS.2019.51. URL : https://doi.org/10.4230/LIPIcs.MFCS.2019.51 (cf. p. 3).

[KL51] S. Kullback et R. A. Leibler. « On Information and Sufficiency ». In : *The Annals of Mathematical Statistics* 22.1 (1951), p. 79-86. DOI : 10.1214/aoms/1177729694. URL : https://doi.org/10.1214/aoms/1177729694 (cf. p. 25).

[LZ91] Jeffrey C Lagarias et Günter M Ziegler. « Bounds for lattice polytopes containing a fixed number of interior points in a sublattice ». In : *Canadian Journal of Mathematics* 43.5 (1991), p. 1022-1035 (cf. p. 28).

[Lec07] Thierry Lecroq. « Fast exact string matching algorithms ». In : *Information Processing Letters* 102.6 (2007), p. 229-235. ISSN : 0020-0190 (cf. p. 66, 67).

[Lel12] M. Lelarge. « A new approach to the orientation of random hypergraphs ». In : *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '12. Kyoto, Japan : SIAM, 2012, p. 251-264. URL : http://dl.acm.org/citation.cfm?id=2095116.2095139 (cf. p. 44).

[LPW08] D.A. Levin, Y. Peres et E.L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Soc., 2008 (cf. p. 16, 25).

[MR92] Heikki Mannila et Kari-Jouko Räihä. « On the complexity of inferring functional dependencies ». In : *Discrete Applied Mathematics* 40.2 (1992), p. 237-243 (cf. p. 43).

[MNW14] Conrado Martínez, Markus Nebel et Sebastian Wild. « Analysis of Branch Misses in Quicksort ». In : (nov. 2014) (cf. p. 3, 64).

[Mar24] Arnaud Mary. *Enumeration of minimal transversals of hypergraphs of bounded VC-dimension*. 2024. arXiv : 2407.00694 [math.CO] (cf. p. 57).

[Moo56] E.F. Moore. « Gedanken experiments on sequential machines ». In : *Automata Studies. Princeton U.* (1956), p. 129-153 (cf. p. 3).

[Mou+09] Alix Mougenot, Alexis Darrasse, Xavier Blanc et Michele Soria. « Uniform random generation of huge metamodel instances ». In : *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2009, p. 130-145 (cf. p. 5).

[Nad89] Denis Naddef. « The Hirsch conjecture is true for (0, 1)-polytopes ». In : *Mathematical Programming : Series A and B* 45.1 (1989), p. 109-110 (cf. p. 28).

[NZ11] Benjamin Nill et Günter M Ziegler. « Projecting lattice polytopes without interior lattice points ». In : *Mathematics of Operations Research* 36.3 (2011), p. 462-467 (cf. p. 28).

[Nis+21] Akio Nishimoto, Noriki Fujisato, Yuto Nakashima et Shunsuke Inenaga. « Position Heaps for Cartesian-tree Matching on Strings and Tries ». In : *SPIRE*. Lille, France, 2021, p. 241-254 (cf. p. 70).

[Ohl13] E. Ohlebusch. *Bioinformatics Algorithms : Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Oldenbusch Verlag, 2013 (cf. p. 71, 78).

[Oiz+22] Tsubasa Oizumi, Takeshi Kai, Takuya Mieno, Shunsuke Inenaga et Hiroki Arimura. « Cartesian Tree Subsequence Matching ». In : *CPM*. Sous la dir. d'Hideo Bannai et Jan Holub. T. 223. LIPIcs. Prague, Czech Republic : Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 14 :1-14 :18 (cf. p. 70).

[Par+19] Sung Park, Amihood Amir, Gad Landau et Kunsoo Park. « Cartesian Tree Matching and Indexing ». In : *CPM*. T. 16. Pisa, Italy, 2019, p. 1-14 (cf. p. 70, 71, 77).

[Par+20] Sung Gwan Park, Magsarjav Bataa, Amihood Amir, Gad M. Landau et Kunsoo Park. « Finding patterns and periods in Cartesian tree matching ». In : *Theoret. Comput. Sci.* 845 (2020), p. 181-197. issn : 0304-3975 (cf. p. 70).

[PT11] Hannu Peltola et Jorma Tarhio. « Variations of Forward-SBNDM ». In : *Stringology*. 2011 (cf. p. 67).

[Rea78] Ronald C Read. « Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations ». In : *Annals of discrete mathematics*. T. 2. Elsevier, 1978, p. 107-120 (cf. p. 43).

[Rém85] Jean-Luc Rémy. « Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire ». In : *RAIRO. Informatique théorique* 19.2 (1985), p. 179-195 (cf. p. 8).

[SS98] Saswati Sarkar et Kumar N Sivarajan. « Hypergraph models for cellular mobile communication systems ». In : *IEEE Transactions on Vehicular Technology* 47.2 (1998), p. 460-471 (cf. p. 43).

[Seb99] András Sebő. « An introduction to empty lattice simplices ». In : *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 1999, p. 400-414 (cf. p. 28).

[SB14] Julian Shun et Guy E. Blelloch. « A Simple Parallel Cartesian Tree Algorithm and Its Application to Parallel Suffix Tree Construction ». In : *ACM Trans. Parallel Comput.* 1.1 (2014), p. 20. issn : 2329-4949 (cf. p. 70).

[Son+21] S. Song, G. Gu, C. Ryu, S. Faro, T. Lecroq et K. Park. « Fast algorithms for single and multiple pattern Cartesian tree matching ». In : *Theoret. Comput. Sci.* 849 (2021), p. 47-63 (cf. p. 70).

[Tak07] Ken Takata. « A Worst-Case Analysis of the Sequential Method to List the Minimal Hitting Sets of a Hypergraph ». In : *SIAM J. Discret. Math.* 21 (4 2007), p. 936-946. issn : 0895-4801 (cf. p. 46).

[THG16] Jorma Tarhio, Jan Holub et Emanuele Giaquinta. « Technology Beats Algorithms (in Exact String Matching) ». In : *Software : Practice and Experience* 47 (déc. 2016) (cf. p. 60).

[Val+09] Brigitte Vallée, Julien Clément, James Allen Fill et Philippe Flajolet. « The Number of Symbol Comparisons in QuickSort and QuickSelect ». In : *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*. Sous la dir. de Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas et Wolfgang Thomas. T. 5555. Lecture Notes in Computer Science. Springer, 2009, p. 750-763. doi : 10.1007/978-3-642-02927-1_62. url : https://doi.org/10.1007/978-3-642-02927-1%5C_62 (cf. p. 3, 18).

[Vui80] Jean Vuillemin. « A Unifying Look at Data Structures ». In : *Commun. ACM* 23.4 (1980), p. 229-239. issn : 0001-0782 (cf. p. 70).