Autour des permutations séparables

Mathilde Bouvel

ENS de Cachan

Stage de Master MPRI

Sous la direction de Dominique Rossin

LIAFA, CNRS et Université Paris 7

Mars à Août 2006

Autour des permutations séparables

Mathilde Bouvel

Mémoire de Master MPRI-2
à la suite du stage effectué au Liafa
sous la direction de Dominique Rossin

Mars à Août 2006

Le travail dont le présent mémoire rend compte se situe au confluent de deux vastes domaines de recherche : la *combinatoire* (en particulier énumérative) et l'*algorithmique* (à travers les problèmes de recherche de sous-structures).

La combinatoire est une branche des mathématiques discrètes qui étudie des collections finies d'objets et dont un but est de compter les éléments dans ces ensembles finis. Les permutations sont un exemple d'objets combinatoires.

En algorithmique, les problèmes de recherche de sous-structures sont très répandus : recherche de motifs dans les mots, recherche de sous-graphes possédant certaines propriétés... Le problème algorithmique qui est au centre de mon travail est la recherche de motif dans les permutations.

Les objets combinatoires auxquels nous allons nous intéresser dans ce qui suit sont les permutations *séparables*. La classe des permutations séparables fait partie des nombreuses classes de permutations *à motifs exclus*. Ce rapport se place dans une dynamique de recherche actuelle étant donné qu'une grande partie des travaux combinatoires récents sur les permutations concernent des classes de permutations à motifs exclus.

Ce mémoire est organisé de la façon suivante. D'abord, une introduction motive et décrit le problème de recherche de motif dans les permutations. Ensuite, je présente les permutations séparables, en en donnant différentes définitions que l'on trouve dans la littérature. Puis j'envisage des questions d'énumération sur les permutations séparables, principalement avec une approche bijective, nouvelle pour cette classe de permutations. Enfin je traite des problèmes algorithmiques de recherche d'un motif séparable dans une permutation quelconque, et de recherche de plus grand motif commun à une permutation séparable et une permutation quelconque. Pour terminer, j'utilise l'outil de la décomposition en intervalles communs des permutations pour donner un résultat sur le problème de recherche de plus grand motif commun dans une classe de permutations plus large que la classe des permutations séparables.

Dans une annexe, j'aborde un tout autre sujet, qui reste cependant au croisement de l'algorithmique et de la combinatoire : les mots bicoloriés.

Table des matières

Table des figures

1 Motivations et présentation du problème

Le travail dont je rends compte dans ce rapport trouve sa motivation principale dans un problème transversal à de nombreux domaines de l'informatique : la recherche de plus grande "sous-structure" commune à deux objets structurés. Un exemple de problème entrant dans ce cadre est la recherche de *plus grand sous-graphe commun* à deux graphes, problème dont les champs d'applications sont nombreux et très divers, allant de la chimie et de la biologie ([30] en est un exemple parmi beaucoup d'autres) au traitement de l'image [32]. Ce problème est malheureusement NP-complet [20], et on s'intéresse donc à des sous-classes de graphes pour lesquelles on espère trouver des algorithmes polynomiaux.

Parmi les sous-problèmes étudiés, la recherche de *plus grand sous-arbre commun* à deux arbres a fait couler beaucoup d'encre. Par exemple [7] et [36] présentent un panorama des résultats connus sur ce thème, et sur le calcul de la *distance d'édition sur les arbres*, ces deux problèmes étant très naturellement reliés. Comme le montre l'algorithme fondateur de Zhang et Shasha en 1995 [39], la recherche de plus grand sous-arbre commun (de même que le calcul de la distance d'édition entre deux arbres) peut être résolue en temps polynomial. Dulucq et Touzet [14] ont montré par la suite que cet algorithme et un autre, décrit par Klein [25], sont en fait des cas particuliers d'une structure générale d'algorithmes calculant un plus grand sous-arbre commun en temps polynomial.

Plus récemment, Anne Micheli et Dominique Rossin ont montré [27] que la recherche de plus grand motif commun (i.e. d'une sous-permutation commune de longueur maximale) à deux permutations *triables par pile* est un problème qui peut être formulé comme recherche de plus grand sous-arbre commun à deux arbres. L'algorithme de Zhang et Shasha fournit alors un algorithme polynomial pour la recherche de plus grand motif commun à deux permutations triables par pile, alors que ce problème est NP-complet dans le cas de deux permutations quelconques [9].

Ce résultat a suscité l'espoir que, dualement au cas précédent, des algorithmes pour la recherche de plus grand motif commun à deux permutations (prises dans une certaine classe un peu plus grande que les permutations triables par pile) fournissent des outils pour rechercher des sous-graphes communs à deux graphes pris dans une classe plus générale que les arbres. Pour nous diriger dans cette voie, nous nous sommes intéressés aux permutations *séparables*, qui non seulement possèdent des propriétés combinatoires intéressantes, mais qui ont aussi été étudiées d'un point de vue algorithmique. On dispose en effet depuis 1996 de l'algorithme de Bose, Buss et Lubiw [9] (amélioré ensuite par Ibarra [23]) qui recherche une occurrence d'un motif séparable dans une permutation quelconque en temps polynomial.

En modifiant un peu l'algorithme de Bose, Buss et Lubiw, nous avons pu donner un algorithme qui calcule, en temps polynomial, un plus grand motif commun à deux permutations dont une est séparable. Cet algorithme a pu en outre être étendu à une classe de permutations plus générales que les séparables, en faisant intervenir une construction associée à la décomposition modulaire de graphes. Dans la suite du rapport, je définis, puis énumère les permutations

séparables, avant de présenter lesdits algorithmes.

Le rapport comporte une dernière partie indépendante qui ne traite pas des permutations mais des mots bicoloriés, sujet qui a occupé le début de mon stage.

2 Généralités sur les permutations séparables

Les permutations séparables sont au coeur du travail que j'ai effectué dans mon stage. Elles ont été introduites dans [9, 23], et on peut en proposer plusieurs définitions équivalentes. Cette section en introduit les définitions "historiques" ainsi que quelques propriétés, et on verra plus loin (théorème 5.6) une caractérisation nouvelle de cette classe de permutations.

2.1 Permutations à motifs exclus

La classe des permutations séparables fait partie des classes de *permutations à motifs exclus*. De nombreuses classes de permutations étudiées dans la littérature peuvent être définies comme des classes de permutations à motifs exclus, même si ce n'est pas toujours ainsi qu'elles sont introduites à l'origine. C'est le cas par exemple des permutations triables par pile [27], des permutations de Schröder [5], des "deque permutations" de Knuth [38], des permutations de Baxter [10], des involutions vexilliaires [10], et de nombreux articles s'attachent à leur étude [2, 16, 37]. Je donne dans ce qui suit les définitions essentielles pour comprendre ce que sont les permutations à motifs exclus.

Dans la plupart des travaux menés en algorithmique et en combinatoire, les permutations sont représentées comme des séquences finies de nombres, telles que dans une séquence de longueur n apparaisse exactement une fois chaque entier entre 1 et n (n est la *taille* de la permutation). Nous écrirons par exemple 145326 la permutation de $\{1, 2, 3, 4, 5, 6\}$ qui à 1 associe 1, à 2 associe 4, à 3, 5, à 4, 3, à 5, 2 et à 6, 6. Cette représentation permet d'employer certains concepts généralement associés aux mots, comme les sous-séquences ou sous-mots. Précisons aussi que, pour tout $n \in \mathbb{N}$, nous notons S_n l'ensemble des permutations de $\{1, 2, 3, \cdots n\}$ et $S = \cup_{n \geq 1} S_n$.

Définition 2.1. *Soient $\pi \in S_n$ et $\tau \in S_k$ deux permutations, avec $n > k$.*
On dit que π évite *τ s'il n'y a pas de sous-séquence $\pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}$ avec $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ qui est isomorphe en ordre à τ, c'est-à-dire telle que les coefficients π_{i_j} soient dans le même ordre que les coefficients de τ.*
Si au contraire une telle sous-séquence existe, on dit que π contient *τ, et que $\pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}$ est une* occurrence *de τ dans π.*
La permutation τ est appelée motif.

Par exemple, la permutation 135624 *évite* le motif 321 mais *contient* le motif 132 (dans la sous-séquence 164 par exemple).

Nous noterons par $S_n(\tau)$ l'ensemble des permutations de S_n qui évitent τ, et $S_n(T)$ l'ensemble des permutations de S_n qui évitent tous les motifs de l'ensemble de motifs T. Pour un ensemble de motifs T donné, $S(T) = \cup_{n \geq 1} S_n(T)$

est une classe de permutations *à motifs exclus*.

Un des résultats les plus importants sur les permutations à motifs exclus est la conjecture de Stanley-Wilf, démontrée en 2004 par Marcus et Tardos [26] :

Théorème 2.2. *Pour tout motif τ, considérons la classe $S(\tau)$ de permutations à motifs exclus. Il existe une constante c_τ telle que pour tout $n \in \mathbb{N}$, $|S_n(\tau)| \leq c_\tau^n$.*

Une classe de permutations à motifs exclus contient donc une très petite proportion de l'ensemble des permutations.

2.2 Définition des permutations séparables

Les permutations séparables peuvent être définies en termes de motifs exclus :

Définition 2.3. *Les permutations séparables sont les permutations évitant les motifs* 2413 *et* 3142 *: $S(2413, 3142)$.*

Il existe une définition alternative, l'équivalence de ces deux définitions étant démontrée dans [9] :

Définition 2.4. *Les permutations séparables sont celles pour lesquelles il existe un arbre de séparation.*

Un arbre de séparation *d'une permutation π de taille n est un arbre binaire plan dont les feuilles sont, dans l'ordre de lecture de gauche à droite, π_1, π_2, ..., π_n, et tel que pour tout noeud V de l'arbre, l'ensemble des feuilles dans le sous-arbre de racine V forme un intervalle.*

Démonstration. Il est clair que pour une permutation qui n'est pas séparable, c'est-à-dire qui contient un motif 2413 ou 3142, il n'existe pas d'arbre de séparation.

Pour la réciproque, on raisonne par récurrence sur la longueur n des permutations séparables.

On peut prendre $n = 4$ pour le cas de base, et on vérifie aisément que toutes les permutations de taille 4 hormis 2413 et 3142 sont séparables au sens de la définition 2.3 et possèdent un arbre de séparation.

Supposons maintenant que toutes les permutations séparables de taille n au sens de la définition 2.3 possèdent un arbre de séparation. Soit π une permutation de taille $n+1$ ne contenant aucun motif 2413 ni 3142. Il suffit de montrer qu'il y a un indice i, $1 \leq i \leq n$ tel que $\{\pi_i, \pi_{i+1}\}$ forme un intervalle. Ainsi, en appliquant l'hypothèse de récurrence à la permutation π' définie par

$$\text{pour } 1 \leq j \leq i : \begin{cases} \pi'_j = \pi_j & \text{si } \pi_j \leq \min(\pi_i, \pi_{i+1}) \\ \pi'_j = \pi_j - 1 & \text{si } \pi_j > \min(\pi_i, \pi_{i+1}) \end{cases}$$
$$\text{pour } i < j \leq n : \begin{cases} \pi'_j = \pi_{j+1} & \text{si } \pi_{j+1} \leq \min(\pi_i, \pi_{i+1}) \\ \pi'_j = \pi_{j+1} - 1 & \text{si } \pi_{j+1} > \min(\pi_i, \pi_{i+1}) \end{cases}$$

on pourra conclure que π possède bien un arbre de séparation. Il suffit pour cela de remplacer, dans l'arbre de séparation de π', la feuille correspondant à π'_i par un noeud interne ayant deux fils qui sont des feuilles d'étiquettes π_i et π_{i+1}, et

de corriger les étiquettes des autres feuilles en conséquence. Remarquons que la permutation π' définie évite bien 2413 et 3142, puisque π évite ces deux motifs, et donc qu'il est licite d'appliquer l'hypothèse de récurrence à π'.

Reste à démontrer l'existence d'un tel indice i.

Supposons que $\pi_1 > \pi_2$, le cas $\pi_1 < \pi_2$ se traitant de manière analogue. Si $\pi_1 = \pi_2 + 1$, l'indice $i = 1$ convient. Sinon, il existe un indice $k > 2$ tel que $\pi_k = \pi_1 - 1$. Définissons aussi j le plus petit indice tel que $R = \{\pi_2, \pi_3, \ldots, \pi_j\}$ contienne toutes les valeurs $\pi_2, \pi_2 + 1, \ldots, \pi_k - 1, \pi_k$. Nous avons donc $\pi_2 < \pi_j \leq \pi_k < \pi_1$.

Si R n'est pas un intervalle, deux cas se présentent :

- soit il existe un indice $\ell \in \{2, \ldots, j\}$ tel que $\pi_\ell > \pi_k$, et donc tel que $\pi_\ell > \pi_1$,
- soit il existe un indice $\ell \in \{2, \ldots, j\}$ et un indice $m > j$ tels que $\pi_\ell < \pi_m < \pi_2$.

Dans le premier cas, $\pi_1\pi_2\pi_\ell\pi_j$ est une occurrence de 3142. Dans le second cas, c'est $\pi_2\pi_\ell\pi_j\pi_m$ qui est une occurrence de 3142. Ceci contredit le fait que π est séparable. On conclut donc que R est un intervalle.

Ainsi, on peut définir la permutation π^R par $\pi^R_p = \pi_{p+1} - \min\{\pi_2, \pi_3, \ldots, \pi_j\} + 1$, pour $1 \leq p \leq j - 1$. Elle est séparable au sens de la définition 2.3. On peut donc lui appliquer l'hypothèse de récurrence : elle possède un arbre de séparation. Une conséquence évidente est alors l'existence d'un indice i entre 2 et $j - 1$ tel que $\{\pi_i, \pi_{i+1}\}$ forme un intervalle. □

À chaque noeud d'un arbre de séparation, on associe donc un intervalle. La définition d'un arbre de séparation implique que chaque noeud interne possède l'un des deux types suivants :

- noeud positif + : si l'intervalle de son fils gauche contient des valeurs supérieures à celles de l'intervalle de son fils droit,
- noeud négatif − : si l'intervalle de son fils gauche contient des valeurs inférieures à celles de l'intervalle de son fils droit,

Remarquons que pour une permutation donnée, il peut exister plusieurs arbres de séparation. La figure 1 représente par exemple deux arbres de séparation possibles pour la permutation $\pi = 856791234$.

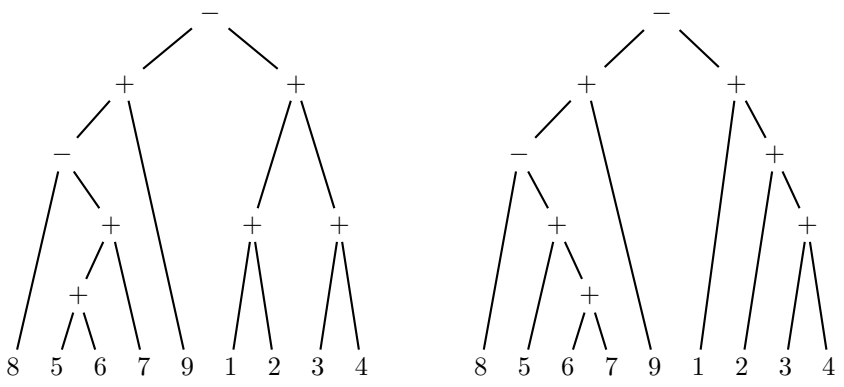


FIG. 1 – Deux arbres de séparation pour $\pi = 856791234$

Une caractérisation en termes de graphes est aussi citée dans [9] :

Définition 2.5. *Soit π une permutation de taille n. Son graphe associé a pour ensemble de sommets $\{1, \ldots, n\}$, et $\{i, j\}$ est une arête (pour $i < j$) si et seulement si i apparaît après j dans π.*

Définition 2.6. *Soit G un graphe. Un P_4 dans G est un chemin* induit *sur 4 sommets.*

Propriété 2.7. *Une permutation est séparable si et seulement si le graphe associé à la permutation ne contient pas de P_4.*

Démonstration. Soit π une permutation, et G son graphe associé.

Supposons d'abord que π n'est pas séparable. π contient donc une occurrence du motif 2413 ou du motif 3142, disons $\pi_i\pi_j\pi_k\pi_l$, avec $i < j < k < l$. Considérons le sous-graphe G' de G induit sur les sommets $\{\pi_i, \pi_j, \pi_k, \pi_l\}$.

Dans le cas d'une occurrence de 2413, les seules arêtes de G' sont, d'après la définition 2.5, $\{\pi_k, \pi_i\}$, $\{\pi_k, \pi_j\}$ et $\{\pi_l, \pi_j\}$. Pour une occurrence de 3142, les arêtes de G' sont exactement $\{\pi_j, \pi_i\}$, $\{\pi_l, \pi_i\}$ et $\{\pi_l, \pi_k\}$.

Comme le montre la figure 2, G' est un P_4 dans ces deux cas. Donc G contient un P_4 induit.

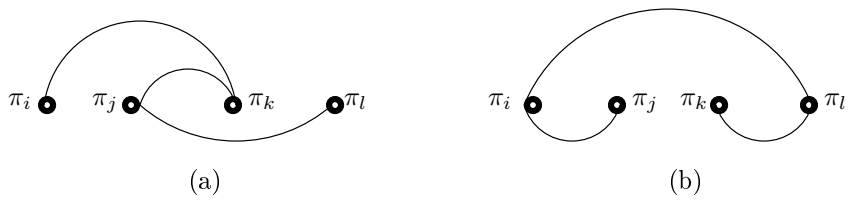


FIG. 2 – Le sous-graphe de G induit sur $\{\pi_i, \pi_j, \pi_k, \pi_l\}$ lorsque $\pi_i\pi_j\pi_k\pi_l$ est une occurrence de (a) 2413 ou de (b) 3142

Supposons à présent que G contient un P_4 induit G', sur les sommets $\{i, j, k, l\}$, les arêtes de G' étant $\{i, j\}$, $\{j, k\}$ et $\{k, l\}$. Par construction de G, la présence de ces trois arêtes signifie que :

(1.a)	$i < j$ et i est après j dans π	ou	(1.b)	$i > j$ et i est avant j dans π
(2.a)	$j < k$ et j est après k dans π	ou	(2.b)	$j > k$ et j est avant k dans π
(3.a)	$k < l$ et k est après l dans π	ou	(3.b)	$k > l$ et k est avant l dans π

En outre, G' étant un P_4 induit, $\{i, k\}$, $\{i, l\}$ et $\{j, l\}$ ne sont pas des arêtes de G. Par définition de G on a encore :

(4)	si $i < k$ alors i est avant k	et	si $i > k$ alors i est après k
(5)	si $i < l$ alors i est avant l	et	si $i > l$ alors i est après l
(6)	si $j < l$ alors j est avant l	et	si $j > l$ alors j est après l

Il suffit alors de raisonner par cas en fonction de qui de $(1.a)$ ou $(1.b)$ (resp. de $(2.a)$ ou $(2.b)$, resp. de $(3.a)$ ou $(3.b)$) est vrai. Dans chacun des huit cas, on arrive à démontrer que les valeurs $\{i, j, k, l\}$ forment une occurrence de 2413 ou de 3142 dans π, ou bien on trouve une contradiction. Le tableau suivant résume les résultats obtenus dans tous les cas :

Formule (1) vraie	Formule (2) vraie	Formule (3) vraie	Motif formé par $\{i,j,k,l\}$ dans π ou contradiction
$(1.a)$	$(2.a)$	$(3.a)$	contradiction avec (5) par ex.
$(1.a)$	$(2.a)$	$(3.b)$	contradiction avec (4)
$(1.a)$	$(2.b)$	$(3.a)$	$\{i,j,k,l\}$ forment un motif 2413 ou 3142
$(1.a)$	$(2.b)$	$(3.b)$	contradiction avec (6)
$(1.b)$	$(2.a)$	$(3.a)$	contradiction avec (6)
$(1.b)$	$(2.a)$	$(3.b)$	$\{i,j,k,l\}$ forment un motif 2413 ou 3142
$(1.b)$	$(2.b)$	$(3.a)$	contradiction avec (4)
$(1.b)$	$(2.b)$	$(3.b)$	contradiction avec (5) par ex.

□

2.3 Arbres de séparation ; arbres de Schröder

Les arbres de séparation tels que définis ci-dessus ont l'avantage d'être des arbres binaires, et donc se prêtent bien à des manipulations algorithmiques. Cependant, comme le montre la figure 1, il n'y a pas unicité de la représentation d'une permutation par un arbre de séparation, ce qui fait que cette structure est peu utilisable d'un point de vue combinatoire, et en particulier énumératif.

On introduit donc un autre type d'arbre de séparation, où les arités des noeuds ne sont plus 2 mais supérieures ou égales à 2. Pour les distinguer, on parlera d'arbres *n-aires* de séparation, par opposition aux arbres *binaires* de séparation. Une permutation sera représentée par un unique arbre n-aire de séparation.

Pour construire l'arbre n-aire de séparation d'une permutation π, on commence par construire un arbre binaire de séparation de π, peu importe lequel. Puis dans cet arbre, on fusionne les noeuds de même signe reliés par une arête, jusqu'à ce que plus aucune fusion ne soit possible (voir figure 3).

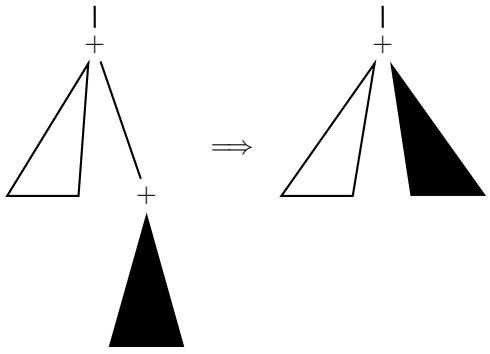


FIG. 3 – Exemple de fusion de deux noeuds positifs

Il est clair que l'arbre ainsi obtenu est unique et ne dépend pas du choix de l'arbre binaire de séparation de π. Il est clair aussi que tous les noeuds

de l'arbre n-aire de séparation représentent des intervalles (constitués par les valeurs des feuilles de leur descendance). Remarquons aussi que si l'on considère une suite de fils consécutifs $V_i, V_{i+1}, \ldots, V_j$ d'un noeud V de l'arbre n-aire de séparation, dont les intervalles associés sont $I_i, I_{i+1}, \ldots, I_j$, alors $\cup_{p=i}^{j} I_p$ est aussi un intervalle. Dans ce qui précède, on ne demande pas que $V_i, V_{i+1}, \ldots, V_j$ représentent l'ensemble des fils de V, mais seulement une partie "en un seul bloc" des fils de V.

En outre, le signe de la racine suffit à déterminer le signe de tous les noeuds internes. En effet, le signe d'un noeud interne qui n'est pas la racine est l'opposé du signe de son père (sinon une fusion serait encore possible), et on peut donc propager ainsi les signes des noeuds depuis la racine vers les feuilles.

Enfin, les arités des noeuds de l'arbre ne peuvent qu'augmenter dans la fusion de noeuds, ce qui fait que les noeuds internes de l'arbre n-aire de séparation ont une arité supérieure ou égale à 2.

La figure 4 donne l'unique arbre n-aire de séparation de la permutation $\pi = 856791234$.

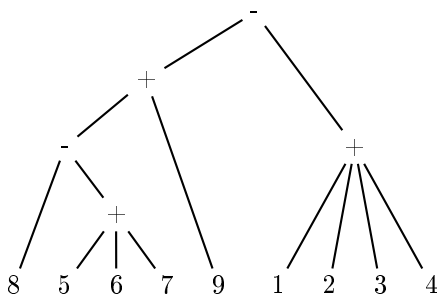


FIG. 4 – L'unique arbre de séparation n-aire pour $\pi = 856791234$

Les arbres plans où l'arité des noeuds internes est supérieure ou égale à 2 sont appelés *arbres de Schröder*. Chaque permutation séparable peut donc être représentée par un arbre de Schröder signé, c'est-à-dire avec un signe pour la racine. Comme nous le verrons dans la partie consacrée à l'énumération des permutations séparables, cette correspondance bijective permet de compter les permutations séparables selon leur taille.

3 Énumération des permutations séparables

Le résultat essentiel concernant l'énumération des permutations séparables est le suivant :

Théorème 3.1. *Le nombre de permutations séparables de taille n est donné par le $(n-1)$-ième nombre de Schröder : s_{n-1}.*

Ce résultat a été démontré par West dans [38] par la méthode des arbres de génération, puis par Ehrenfeucht, Harju, ten Pas et Rozenberg [15].

Dans ce qui suit, je présente d'abord les nombres de Schröder, puis je propose une preuve bijective du théorème 3.1.

3.1 Nombres de Schröder ; chemins de Schröder

Il existe en fait deux suites de nombres de Schröder : les grands nombres de Schröder (s_n) et les petits nombres de Schröder (r_n). Les uns sont liés aux autres par la relation $2r_n = s_n$ pour tout $n \geq 1$. Une preuve bijective de cette relation est donnée dans [31].

Grands nombres de Schröder La série génératrice des grands nombres de Schröder est [33] :

$$S(x) = \sum_{n \geq 0} s_n x^n = \frac{1 - x - \sqrt{x^2 - 6x + 1}}{2x} \tag{1}$$

Les premiers termes de la suite $(s_n)_{n \geq 0}$ sont [33] :

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098$$

Dans ce rapport, lorsque j'écris nombres de Schröder sans plus de précision, je fais référence aux grands nombres de Schröder.

Les nombres de Schröder énumèrent plusieurs classes d'objets intéressantes. On peut par exemple définir s_n comme étant le nombre de chemins de Schröder de taille n.

Définition 3.2. *Un chemin de Schröder de taille n est un chemin dans le quart de plan positif, commençant en $(0,0)$ et arrivant en $(2n,0)$, faisant des pas montants a, de coordonnées $(1,1)$, des pas descendants $\overline{a}$, de coordonnées $(1,-1)$ et des "doubles" pas horizontaux b, de coordonnées $(2,0)$.*

Un exemple de chemin de Schröder est représenté figure 5.

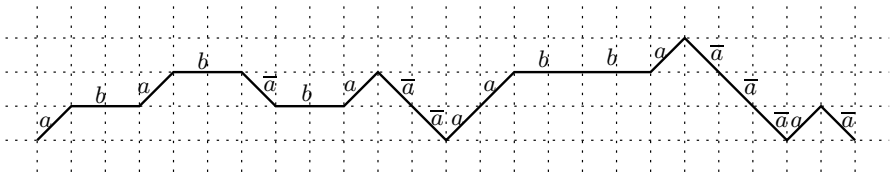


FIG. 5 – Un chemin de Schröder de taille 12

Il est facile de retrouver la série génératrice des nombres de Schröder à partir de la structure des chemins de Schröder. En effet, un chemin de Schröder est

– soit le chemin vide (de taille 0),

– soit commence par un pas montant, et donc est de la forme $aS_1\overline{a}S_2$,

– soit commence par un pas horizontal, et donc est de la forme bS_1,

où S_1 et S_2 sont des chemins de Schröder plus petits.
La série $S(x) = \sum s_n x^n$ vérifie donc l'équation

$$S(x) = 1 + xS^2(x) + xS(x)$$

dont la résolution redonne bien la forme (1).

Petits nombres de Schröder Le n-ième petit nombre de Schröder r_n est défini comme le nombre de parenthésages généraux d'un mot à $n+1$ lettres $z_1 z_2 \cdots z_{n+1}$.

Définition 3.3. *Un parenthésage général de $z_1 z_2 \cdots z_n$ est un parenthésage légal de $z_1 z_2 \cdots z_n$, où chaque lettre correspond à un sous-parenthésage (les parenthèses sont omises), et où chaque paire de parenthèses entoure au moins deux sous-parenthésages.*

Par exemple, les parenthésages généraux de $z_1 z_2 \cdots z_{n+1}$ pour $n = 0, 1, 2$ et 3 sont :

- z_1
- $(z_1 z_2)$
- $(z_1 z_2 z_3)$, $((z_1 z_2) z_3)$, $(z_1 (z_2 z_3))$
- $(z_1 z_2 z_3 z_4)$, $(z_1 (z_2 z_3 z_4))$, $((z_1 z_2 z_3) z_4)$, $((z_1 z_2) z_3 z_4)$, $(z_1 (z_2 z_3) z_4)$, $(z_1 z_2 (z_3 z_4))$, $((z_1 z_2)(z_3 z_4))$, $(z_1 (z_2 (z_3 z_4)))$, $(((z_1 z_2) z_3) z_4)$, $((z_1 (z_2 z_3)) z_4)$, $(z_1 ((z_2 z_3) z_4))$.

J'explicite dans le paragraphe suivant la bijection esquissée plus haut entre les permutations séparables et les arbres de Schröder signés. Puis j'établis une bijection entre les arbres de Schröder (non-signés) et les parenthésages généraux. Enfin, je décris une bijection entre les chemins de Schröder sans pas horizontaux sur l'axe des abscisses et les arbres de Schröder (non-signés). Le point important pour l'énumération est que toutes ces bijections conservent la taille.

3.2 Bijection entre permutations séparables et arbres de Schröder signés

J'ai déjà décrit plus haut comment construire un arbre de Schröder signé à n feuilles à partir d'une permutation séparable de longueur n. L'arbre défini ainsi étant unique, on a donc une fonction des permutations séparables vers les arbres de Schröder signés, qui envoie une permutation de taille n vers un arbre à n feuilles. Il suffit donc de trouver l'inverse de cette fonction pour démontrer qu'elle est une bijection.

Considérons un arbre de Schröder signé à n feuilles. On commence par mettre un signe sur chaque noeud interne, de sorte qu'un noeud et son père aient des signes opposés. L'intervalle $\{1, \ldots, n\}$ est associé à la racine, puis des intervalles sont associés aux fils d'un noeud ayant déjà un intervalle de telle sorte que :

- la largeur de l'intervalle associé au noeud V correspond au nombre de feuilles dans le sous-arbre de racine V,
- si un noeud est positif (resp. négatif), les intervalles correspondant à ses fils lus de gauche à droite sont classés par ordre de valeurs croissantes (resp. décroissantes).

À chaque feuille correspond donc un nombre entre 1 et n, et la lecture des feuilles de l'arbre de gauche à droite donne une permutation séparable (l'arbre donné au départ en est l'arbre n-aire de séparation).

On peut donc énoncer la propriété suivante :

Propriété 3.4. *Les arbres de Schröder signés à n feuilles sont en bijection avec les permutations séparables de taille n.*

Le nombre de permutations séparables de taille n est donc égal à deux fois celui d'arbres de Schröder (non-signés) à n feuilles.

Pour achever de démontrer le théorème 3.1, il suffit donc de démontrer que le nombre d'arbres de Schröder à n feuilles est $\frac{1}{2}s_{n-1} = r_{n-1}$. On propose une preuve bijective de ce résultat dans le paragraphe suivant.

3.3 Bijection entre les arbres de Schröder et les parenthésages généraux

Il est aisé de décrire une bijection entre les parenthésages généraux de mots de n lettres et les arbres de Schröder à n feuilles.

La construction d'un parenthésage général à partir d'un arbre de Schröder est récursive :
- à une feuille on associe une lettre,
- à un noeud interne, qui a par définition $p \geq 2$ fils, notés $V_1, \ldots, V_p$, on associe le parenthésage $(M_1 \ldots M_p)$, où chaque M_i est le parenthésage associé récursivement au noeud V_i de l'arbre.

Le parenthésage associé à un arbre est celui associé à sa racine.

De même, une construction récursive associe à un parenthésage général un arbre de Schröder, en envoyant le nombre de lettres du parenthésage vers le nombre de feuilles de l'arbre :
- à une lettre on associe une feuille,
- à un parenthésage $(M_1 \ldots M_p)$, avec par définition $p \geq 2$, on associe un arbre de racine V ayant p fils, $V_1 \ldots V_p$ de gauche à droite, chaque V_i étant la racine de l'arbre associé récursivement au sous-parenthésage M_i.

Ces deux constructions sont clairement réciproques l'une de l'autre et définissent donc une bijection, illustrée sur un exemple en figure 6.

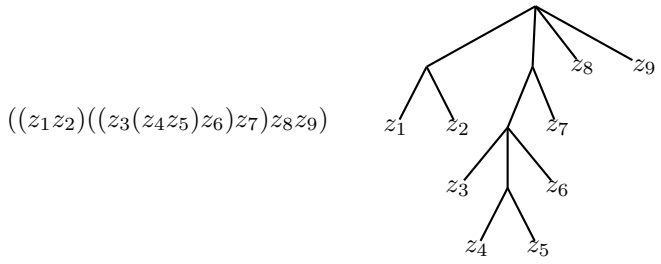


FIG. 6 – Un parenthésage général et un arbre de Schröder qui sont mis en relation par la bijection décrite

On peut alors conclure en énonçant la propriété suivante :

Propriété 3.5. *Les arbres de Schröder à n feuilles sont en bijection avec les parenthésages généraux de mots de n lettres. Il y a donc r_{n-1} arbres de Schröder à n feuilles.*

Le théorème 3.1 est une conséquence directe des propriétés 3.4 et 3.5.

3.4 Bijection entre les arbres de Schröder signés et les chemins de Schröder

Une manière peut-être plus directe pour démontrer que les permutations séparables sont énumérées par la suite des nombres de Schröder est de trouver une bijection entre les arbres de Schröder signés et les chemins de Schröder.

Ce qu'on propose de faire dans ce paragraphe est de partitionner les chemins de Schröder de taille n en deux parties de même cardinalité (ces deux parties pouvant être mises en bijection de manière très naturelle par Φ_1), et d'exhiber une bijection Φ_2 entre les arbres de Schröder non-signés et une de ces parties. Ainsi, on pourra définir une bijection Φ entre les arbres de Schröder signés et les chemins de Schröder par :

– si le signe de T est positif, alors $\Phi(T) = \Phi_2(\overline{T})$,

– si le signe de T est négatif, alors $\Phi(T) = \Phi_1(\Phi_2(\overline{T}))$,

$\overline{T}$ étant l'arbre non-signé associé à l'arbre signé T.

Les deux propriétés suivantes exhibent les bijections Φ_1 et Φ_2.

Propriété 3.6. *Il existe une bijection Φ_1, qui préserve la taille, entre les chemins de Schröder sans pas horizontaux sur l'axe des abscisses d'une part, et les chemins de Schröder avec au moins un pas horizontal sur l'axe des abscisses d'autre part.*

Démonstration. Considérons un chemin de Schröder S avec au moins un pas horizontal sur l'axe des abscisses. On décompose S en $S_1 b S_2$, où b est le pas horizontal sur l'axe des abscisses qui se trouve le plus à droite dans S. Dans S_2, il n'y a donc pas de pas horizontal sur l'axe des abscisses. On définit alors $\Psi_1(S) = aS_1\overline{a}S_2$. $\Psi_1(S)$ est un chemin de Schröder sans pas horizontal sur l'axe des abscisses et de même taille que S.

Réciproquement, considérons un chemin de Schröder S avec aucun pas horizontal sur l'axe des abscisses. S commence nécessairement par un a, et il existe un premier $\overline{a}$ qui revient sur l'axe des abscisses : $S = aS_1\overline{a}S_2$. On définit alors $\Phi_1(S) = S_1 b S_2$. $\Phi_1(S)$ est un chemin de Schröder avec au moins un pas horizontal sur l'axe des abscisses et de même taille que S. En outre, on remarque que Ψ_1 est la fonction réciproque de Φ_1.

La fonction Φ_1 ainsi définie est donc une bijection qui vérifie la propriété annoncée. □

Propriété 3.7. *Il existe une bijection Φ_2 entre les chemins de Schröder de taille $n-1$ sans pas horizontaux sur l'axe des abscisses et les arbres de Schröder non-signés à n feuilles.*

Démonstration. Soit T un arbre de Schröder à n feuilles. On construit un chemin de Schröder en effectuant un parcours en profondeur de T, en commençant toujours par les arêtes le plus à gauche possible, et en écrivant un a lorsque

l'arête parcourue est la plus à gauche dans sa fratrie, un $\overline{a}$ lorsqu'elle est la plus à droite, et un b dans les autres cas.

S est bien un chemin de Schröder puisque S contient autant de a que de $\overline{a}$, et que dans tout préfixe de S, le nombre de a est supérieur ou égal au nombre de $\overline{a}$. On remarque aussi que par construction S n'a pas de pas horizontal sur l'axe des abscisses.

Reste à calculer la taille de S. Notons m le double de la taille de S, c'est-à-dire l'abscisse du dernier point de S. Le traitement de chaque noeud interne v lors du parcours en profondeur de T fait augmenter m de $2(a(v)-2)+2 = 2(a(v)-1)$, $a(v)$ désignant le nombre de fils de v. Ceci provient du fait que toutes les arêtes sortant de v sont codées par un b, augmentant l'abscisse de 2, sauf la première et la dernière, codées respectivement par un a et par un $\overline{a}$, augmentant chacun l'abscisse de 1.

La taille de S est donc donnée par :

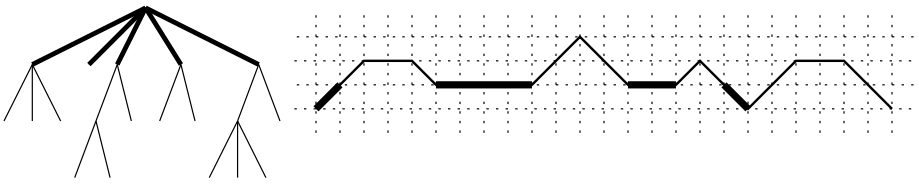
$$\begin{aligned}
\tfrac{m}{2} &= \textstyle\sum_{v \text{ noeud interne}} (a(v)-1) \\
&= \textstyle\sum_{v \text{ noeud interne}} a(v) - \text{nombre de noeuds internes} \\
&= \text{nombre d'arêtes} - \text{nombre de noeuds internes} \\
&= \text{nombre de noeuds} - 1 - \text{nombre de noeuds internes} \\
&= \text{nombre de feuilles} - 1
\end{aligned}$$

La fonction Φ_2 associe à tout arbre de Schröder T à n feuilles le chemin de Schröder S de taille $n-1$, sans pas horizontal sur l'axe des abscisses, comme décrit précédemment.

Il est facile de voir que cette construction est réversible : étant donné un chemin de Schröder S sans pas horizontal sur l'axe des abscisses, on peut reconstruire un arbre de Schröder T dont S est le parcours en profondeur selon le codage décrit plus haut. Il suffit pour cela de décomposer S en $aS_1bS_2bS_3 \dots bS_{p-1}\overline{a}S_p$, où tous les S_i sont des chemins de Schröder sans pas horizontal sur l'axe des abscisses, et tous les b servant dans la décomposition sont à hauteur 1. Par récurrence, des arbres T_i sont associés aux chemins S_i, et T est l'arbre dont la racine a p fils $t_1, \dots, t_p$, t_i étant la racine de T_i.

Φ_2 est donc bien une bijection vérifiant les propriétés annoncées. □

La figure 7 présente un exemple de la construction décrite dans la preuve de la proposition 3.7.



(a) (b)

FIG. 7 – Un arbre de Schröder à 13 feuilles (a) et son chemin associé (b). Les pas utiles à la décomposition en $aS_1bS_2bS_3 \dots bS_{p-1}\overline{a}S_p$ apparaissent en gras.

4 Algorithme de recherche d'un plus grand motif commun à deux permutations, dont une séparable

Dans cette partie du rapport, on s'intéresse à la résolution de problèmes algorithmiques sur les permutations séparables. Un premier problème intermédiaire est la recherche d'une occurrence d'un motif séparable dans une permutation quelconque. Ce problème a été résolu par Bose, Buss et Lubiw [9] et par Ibarra [23], et je présente leurs résultats. Le second problème que je développerai est la recherche d'un plus grand motif commun à une permutation séparable et une permutation quelconque, un objectif dans le stage étant de concevoir des algorithmes trouvant des plus grands motifs communs entre deux permutations prises dans des classes bien choisies. Comme le montre l'étude de complexité faite dans le paragraphe suivant, peu de perspectives sont ouvertes concernant les algorithmes pour résoudre le problème général de recherche de plus grand motif commun à deux permutations quelconques.

4.1 Complexité du problème général de recherche d'un plus grand motif commun

Intéressons nous d'abord au problème de recherche d'une occurrence d'un motif τ de taille k dans une permutation π de longueur n, aussi appelé problème de "pattern matching". La permutation π est parfois appelée *texte* dans ce contexte. Je rappelle que ce problème consiste à déterminer une sous-séquence $\pi_{i_1}\pi_{i_2}\cdots\pi_{i_k}$ de π, avec $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, telle que l'ordre relatif des éléments de $\pi_{i_1}\pi_{i_2}\cdots\pi_{i_k}$ entre eux soit donné par le motif τ : $\pi_{i_j} < \pi_{i_\ell}$ si et seulement si $\tau_j < \tau_\ell$.

Le problème de décision associé est de savoir, étant donnés π et τ, s'il existe une occurrence de τ dans π. Ce problème est NP-complet. En effet, il est clairement dans NP, en choisissant pour témoin les indices $i_1, i_2, \cdots, i_k$. Et pour prouver qu'il est NP-dur, Bose, Buss et Lubiw ont démontré [9] que $3SAT$ se réduit à ce problème.

Considérons maintenant le problème de recherche de plus grand motif commun à deux permutations. Il s'agit, étant données deux permutations π_1 et π_2, de trouver un motif τ le plus long possible tel que π_1 et π_2 contiennent τ. Le problème de "pattern matching" se réduit facilement à ce problème puisque, si un algorithme calcule un plus grand motif commun m à π_1 et π_2, il suffit de tester si $m = \pi_1$ pour décider si oui ou non il existe une occurrence de π_1 dans π_2. Remarquons qu'en toute généralité, il peut exister plusieurs plus grands motifs communs distincts à π_1 et π_2, mais qu'il est unique (et égal à π_1) lorsque π_2 contient π_1.

Une conséquence de la NP-complétude du "pattern matching" est donc le théorème suivant :

Théorème 4.1. *Le problème de recherche de plus grand motif commun à deux permutations quelconques est NP-dur.*

Une orientation de recherche est de déterminer dans quelle classe de complexité exactement se trouve le problème de recherche de plus grand motif com-

mun à deux permutations, mais ce n'est pas ce point de vue que nous avons adopté pendant mon stage. Nous nous sommes plutôt intéressés à la recherche d'algorithmes en temps polynomial pour les problèmes de "pattern matching" et de plus grand motif commun, restreints à des sous-classes de l'ensemble des permutations : en particulier à la classe des permutations séparables.

4.2 Recherche d'une occurrence d'un motif séparable dans une permutation générale

Des algorithmes polynomiaux ont été proposés pour résoudre ce problème : celui de Bose, Buss et Lubiw [9] a une complexité en temps $\mathcal{O}(kn^6)$ (k étant la taille du motif et n la taille de la permutation texte). Cette complexité descend à $\mathcal{O}(kn^4)$ dans l'algorithme d'Ibarra [23]. Dans le premier cas, la complexité en espace est $\mathcal{O}(kn^4)$, contre $\mathcal{O}(kn^3)$ dans le deuxième cas.

Ces deux algorithmes utilisent la programmation dynamique. Dans le premier, il est facile de comprendre le fonctionnement et la correction de l'algorithme. Dans le second, bien que les idées essentielles soient les mêmes, le gain en complexité n'est obtenu qu'au prix de preuves de correction plus élaborées.

Les deux algorithmes utilisent la structure d'arbre de séparation binaire du motif séparable. Je commence donc par décrire un algorithme qui prend en entrée un motif, et qui retourne un arbre de séparation du motif s'il est séparable, et une erreur sinon.

Dans toute cette partie, je considère un motif séparable τ de taille k et une permutation générale π de taille n.

4.2.1 Algorithme de calcul d'un arbre de séparation

Bose, Buss et Lubiw font une description assez "haut-niveau" d'un algorithme en temps linéaire qui teste si une permutation est séparable, et qui calcule dans

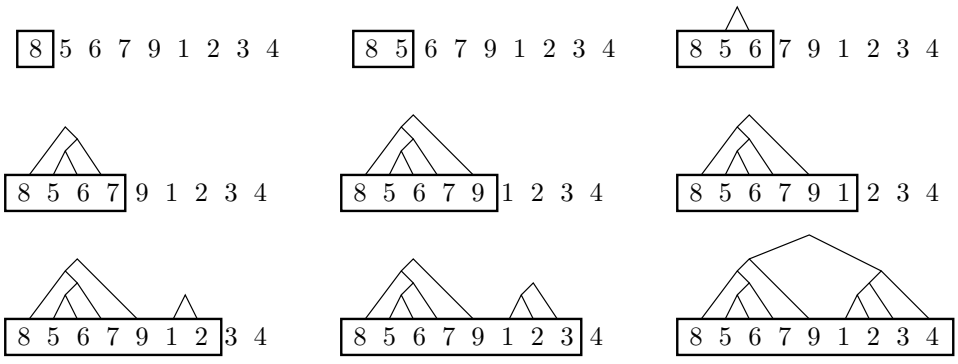


FIG. 8 – Calcul par l'algorithme 1 d'un arbre de séparation de $\pi = 856791234$. Les encadrés représentent la partie de la permutation qui a été traitée jusqu'à cette étape de l'algorithme.

l'affirmative un arbre de séparation binaire de la permutation donnée en entrée. J'en donne dans l'algorithme 1 une description plus formelle. J'ai aussi programmé cet algorithme en Java.

L'idée est de parcourir linéairement la permutation de gauche à droite, pour construire au fur et à mesure des sous-arbres de l'arbre de séparation, et en fusionnant dès que c'est possible ces sous-arbres avec ceux qui sont juste avant eux dans la pile des arbres construits dans les étapes précédentes de l'algorithme (directement à leur gauche sur la figure 8). Un exemple du déroulement de l'algorithme est donné figure 8.

Par les valeurs d'un arbre, j'entends les valeurs de ses feuilles. Tous les arbres considérés dans l'algorithme sont des sous-arbres d'un arbre de séparation, et leurs valeurs constituent donc des intervalles. Pour un arbre T, je note $Min(T)$ et $Max(T)$ ses valeurs minimale et maximale respectivement.

Algorithm 1 L'algorithme de calcul de l'arbre de séparation d'une permutation

```
INPUT : A permutation π of length n
stack ← an empty tree stack
Push a tree consisting of one node labelled π_1 on stack
for i=2 to n do
  if π_i ≠ Min(top of stack) − 1 and π_i ≠ Max(top of stack) + 1 then
    Push a tree consisting of one node labelled π_i on stack
  else
    next_top ← a tree consisting of one node labelled π_i
    while stack is not empty and
        { Min(next_top) = Max(top of stack) + 1 or
          Max(next_top) = Min(top of stack) − 1 } do
      if Min(next_top) = Max(top of stack) + 1 then
        next_top ← a new tree with root labelled + which has the top of
        stack as a left subtree (removed from stack) and next_top as a
        right subtree
      else
        if Max(next_top) = Min(top of stack) − 1 then
          next_top ← a new tree with root labelled − which has the top of
          stack as a left subtree (removed from stack) and next_top as a
          right subtree
        end if
      end if
    end while
    Push next_top on stack
  end if
end for
separating_tree ← the top of stack, removed from stack
if stack is empty then
  Return separating_tree
else
  Return an exception : π is not separable
end if
```

Cet algorithme est implicitement utilisé dans les deux algorithmes des paragraphes suivants. En effet, l'algorithme de Bose, Buss et Lubiw aussi bien que celui d'Ibarra utilise la structure d'arbre de séparation d'un motif séparable. Il y a donc un préalable à ces algorithmes qui est d'utiliser l'algorithme 1 sur le motif donné en entrée. Ce précalcul est en temps linéaire, donc est négligeable par rapport à la complexité des algorithmes de Bose, Buss et Lubiw, et d'Ibarra.

4.2.2 Algorithme de Bose, Buss et Lubiw

L'idée clé de cet algorithme est de chercher une occurrence du motif séparable τ dans la permutation π en suivant la structure de l'arbre de séparation de τ.
Notons T un arbre binaire de séparation de τ, $T(V)$ le sous-arbre de T de racine V pour un noeud V de T, et $P(T(V))$ le sous-motif de τ dont $T(V)$ est un arbre de séparation.

Soit alors un noeud interne V de T, ayant deux fils V_L et V_R. Dans le cas où V est positif (resp. négatif), s'il existe deux entiers h et c tels que
- il existe une occurrence de $P(T(V_L))$ dans $\pi_1 \dots \pi_{h-1}$ n'utilisant que des éléments de π de valeur comprise entre 1 et $c-1$ (resp. entre c et n),
- et il existe une occurrence de $P(T(V_R))$ dans $\pi_h \dots \pi_n$ n'utilisant que des éléments de π de valeur comprise entre c et n (resp. entre 1 et $c-1$),

alors, en les concaténant, on obtient une occurrence de $P(T(V))$ dans π. Et il est facile de voir que cette implication est en fait une équivalence.

Pour exploiter cette idée dans tout l'arbre, on maintient pour chaque noeud V de l'arbre un tableau $M(V, \_, \_, \_, \_)$ à quatre dimensions tel que $M(V, i, j, a, b)$ contient 1 s'il existe une occurrence de $P(T(V))$ dans $\pi_i \pi_{i+1} \cdots \pi_j$ n'utilisant que des éléments de π de valeur comprise entre a et b ($i \leq j$ et $a \leq b$), et 0 sinon.

Il est alors possible de remplir les tableaux $M(V, \_, \_, \_, \_)$ en commençant par les feuilles puis en remontant vers la racine de la façon suivante.

Pour un noeud V de T qui est une feuille, $M(V, i, j, a, b) = 1$ si et seulement s'il existe $h \in \{i, i+1, \dots, j\}$ tel que $a \leq \pi_h \leq b$.

Pour un noeud interne V de T ayant pour fils V_L et V_R, on a :
- si V est positif, alors $M(V, i, j, a, b) =$
  $Max\{M(V_L, i, h-1, a, c-1) \cdot M(V_R, h, j, c, b) : i < h \leq j, a < c \leq b\}$
- si V est négatif, alors $M(V, i, j, a, b) =$
  $Max\{M(V_L, i, h-1, c, b) \cdot M(V_R, h, j, a, c-1) : i < h \leq j, a < c \leq b\}$

Le calcul de $M(V, i, j, a, b)$ à partir des valeurs précédentes pour un noeud interne V se fait donc en temps $\mathcal{O}(n^2)$. Par conséquent, pour chaque noeud interne V, pour remplir tout le tableau $M(V, \_, \_, \_, \_)$, il faut un temps $\mathcal{O}(n^6)$, donc un temps $\mathcal{O}(kn^6)$ pour l'ensemble des noeuds internes. L'espace occupé par l'ensemble des tableaux est quant à lui $\mathcal{O}(kn^4)$. Remarquons que l'étape d'initialisation (remplir les tableaux associés aux feuilles) peut être naïvement réalisée en temps $\mathcal{O}(kn^5)$. La complexité en temps totale pour l'algorithme de Bose, Buss et Lubiw est donc $\mathcal{O}(kn^6)$.

4.2.3 Algorithme d'Ibarra

Pour décider l'existence d'une occurrence de τ dans π, l'algorithme d'Ibarra utilise comme le précédent des techniques de programmation dynamique. Mais pour gagner en complexité par rapport à l'algorithme de Bose, Buss et Lubiw, il considère moins de sous-problèmes par noeud interne, et le calcul des valeurs pour un noeud interne à partir des valeurs de ses fils se fait plus rapidement que dans l'algorithme de Bose, Buss et Lubiw.

Pour chaque noeud V de l'arbre de séparation T de τ, l'algorithme d'Ibarra ne construit pas un mais deux tableaux, qui sont en revanche de dimension 3 au lieu de 4 : $L(V, \_, \_, \_)$ et $H(V, \_, \_, \_)$. En utilisant les mêmes notations que dans le paragraphe précédent, ces tableaux sont définis par : pour tout noeud V de T, pour tout $1 \leq i \leq j \leq n$ et pour tout $1 \leq x \leq n$

- $L(V, i, j, x) = Max\{\{0\} \cup \{y :$ il y a une occurrence de P(T(V)) dans $\pi_i \pi_{i+1} \cdots \pi_j$ dont le plus petit élément est y et le plus grand élément est $\leq x\}\}$
- $H(V, i, j, x) = Min\{\{n+1\} \cup \{y :$ il y a une occurrence de P(T(V)) dans $\pi_i \pi_{i+1} \cdots \pi_j$ dont le plus grand élément est y et le plus petit élément est $\geq x\}\}$

En particulier, $L(V, i, j, x) > 0$ si et seulement s'il existe une occurrence de $P(T(V))$ dans $\pi_i \pi_{i+1} \cdots \pi_j$ dont les éléments sont inférieurs ou égaux à x, et $H(V, i, j, x) < n + 1$ si et seulement s'il existe une occurrence de $P(T(V))$ dans $\pi_i \pi_{i+1} \cdots \pi_j$ dont les éléments sont supérieurs ou égaux à x.

Il est important de remarquer que $L(V, i, j, x)$ ne peut pas décroître lorsque i diminue, ou lorsque j augmente, ou lorsque x augmente. De même, $H(V, i, j, x)$ ne peut pas croître lorsque i diminue, ou lorsque j augmente, ou lorsque x diminue. Ces remarques sont essentielles pour la démonstration de la correction de l'algorithme d'Ibarra.

Je ne détaille pas ici la structure de l'algorithme. L'espace occupé par l'ensemble des tableaux $L(V, \_, \_, \_)$ et $H(V, \_, \_, \_)$ est clairement $\mathcal{O}(kn^3)$. Il est aussi facile de voir que pour une feuille V de T, on peut remplir toutes les cases des tableaux $L(V, \_, \_, \_)$ et $H(V, \_, \_, \_)$ en temps $\mathcal{O}(n^4)$. Dans [23], Ibarra donne une procédure qui permet de remplir les deux tableaux $L(V, \_, \_, \_)$ et $H(V, \_, \_, \_)$ pour un noeud interne V ayant deux fils V_L et V_R en temps $\mathcal{O}(n^4)$ à partir des tableaux $L(V_L, \_, \_, \_)$, $H(V_L, \_, \_, \_)$, $L(V_R, \_, \_, \_)$ et $H(V_R, \_, \_, \_)$. Il démontre aussi la correction de cette procédure.

En remarquant qu'il existe une occurrence de τ dans π si et seulement si $L(R, 1, n, n) > 0$ si et seulement si $H(R, 1, n, 1) < n + 1$, R étant la racine de T, on obtient donc un algorithme qui décide s'il existe une occurrence de τ dans π en temps $\mathcal{O}(kn^4)$ et en espace $\mathcal{O}(kn^3)$.

4.3 Recherche d'un plus grand motif commun à une permutation séparable et une permutation générale

Le problème qui est au centre de mon stage n'est pas celui du "pattern matching", pour lesquels les algorithmes de Bose, Buss et Lubiw, et d'Ibarra fournissent des solutions polynomiales lorsque le motif dont on recherche une

occurrence est séparable. C'est, si l'on veut, une généralisation de ce problème : la recherche de plus grand motif commun à deux permutations. J'ai expliqué plus haut que ce problème est NP-dur, si l'on n'impose pas de restriction sur les permutations. C'est pourquoi on s'est intéressé à la recherche de plus grand motif commun à deux permutations séparables, pour laquelle on espérait pouvoir trouver un algorithme polynomial.

En fait, il est possible de faire encore mieux en temps polynomial : trouver un plus grand motif commun à une permutation séparable et une permutation quelconque. En effet, on s'est aperçu que l'algorithme de Bose, Buss et Lubiw pouvait être légèrement modifié afin de résoudre ce problème. En utilisant l'algorithme 1, j'ai programmé en Java cette adaptation de l'algorithme de Bose, Buss et Lubiw, que je décris dans la suite.

Considérons deux permutations π et π', telle que π est séparable, et dont on cherche un plus grand motif commun. On note n la taille de π et n' la taille de π'. Comme dans l'algorithme de Bose, Buss et Lubiw, on utilise un ensemble de tableaux $M(V, \_, \_, \_, \_)$, un pour chaque noeud V d'un arbre binaire de séparation T de π. T peut par exemple être donné par l'algorithme 1.

Comme précédemment, on note $T(V)$ le sous-arbre de T de racine V pour un noeud V de T, et $P(T(V))$ la sous-permutation de π dont $T(V)$ est un arbre de séparation. $M(V, i, j, a, b)$ contiendra un plus grand motif commun à $P(T(V))$ et $\pi'_i\pi'_{i+1}\cdots\pi'_j$, dont l'occurrence dans $\pi'_i\pi'_{i+1}\cdots\pi'_j$ n'utilise que des valeurs entre a et b, pour $1 \leq i \leq j \leq n'$ et $1 \leq a \leq b \leq n'$. Si $i > j$ ou $a > b$, $M(V, i, j, a, b)$ est le motif vide.

On peut remarquer qu'il n'y a pas unicité du plus grand motif commun à deux permutations. C'est pourquoi l'algorithme que je propose permet de calculer le plus grand motif commun à π et π' qui est le plus petit dans l'ordre lexicographique. Ce critère se transmettant facilement des fils à leur père dans l'algorithme de programmation dynamique, on conserve ainsi sans plus d'effort une certaine régularité quant au motif retourné en sortie par l'algorithme.

En revanche, au niveau de la complexité, si l'on choisit toujours le motif le plus petit dans l'ordre lexicographique parmi ceux de longueur maximale, on est un peu moins efficace : un facteur $\mathcal{O}(min(n, n'))$ vient s'ajouter pour effectuer les comparaisons lexicographiques entre les motifs de longueur maximale. Ce qui nous intéresse étant de trouver un algorithme polynomial, cette remarque n'est pas essentielle pour nous.

Avant de passer à la description de l'algorithme à proprement parler, j'ai besoin de définir deux opérations de concaténation sur les motifs : la concaténation positive $\oplus$ et la concaténation négative $\ominus$.
Considérons deux motifs p et p' de tailles respectives k et k'. On définit

$$p \oplus p' = p_1 \cdots p_k(p'_1 + k) \cdots (p'_{k'} + k) \text{ et}$$

$$p \ominus p' = (p_1 + k') \cdots (p_k + k')p'_1 \cdots p'_{k'}$$

L'algorithme utilise encore la programmation dynamique.

- Pour un noeud V de T qui est une feuille, $M(V, i, j, a, b)$ est le motif 1 s'il existe $h \in \{i, i+1, \ldots, j\}$ tel que $a \leq \pi'_h \leq b$, le motif vide sinon.

- Pour un noeud interne positif V de T, ayant pour fils V_L et V_R, $M(V, i, j, a, b)$ est le plus petit motif pour l'ordre lexicographique parmi les motifs de longueur maximale dans l'ensemble $\{M(V_L, i, j, a, b), M(V_R, i, j, a, b)\} \cup \{M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b) : i < h \leq j, a < c \leq b\}$.
- Pour un noeud interne négatif V de T, ayant pour fils V_L et V_R, $M(V, i, j, a, b)$ est le plus petit motif pour l'ordre lexicographique parmi les motifs de longueur maximale dans l'ensemble $\{M(V_L, i, j, a, b), M(V_R, i, j, a, b)\} \cup \{M(V_L, i, h-1, c, b) \ominus M(V_R, h, j, a, c-1) : i < h \leq j, a < c \leq b\}$.

Une fois les tableaux $M(V, \_, \_, \_, \_)$ remplis, il suffit de retourner le contenu de la case $M(R, 1, n', 1, n')$, R étant la racine de T : c'est le plus grand motif commun à π et π' qui est le plus petit dans l'ordre lexicographique.

Propriété 4.2. *L'algorithme décrit ci-dessus est correct : il retourne bien le plus grand motif commun à π et π' qui est le plus petit dans l'ordre lexicographique.*

Démonstration. La preuve se fait par récurrence. On montre que $M(V, i, j, a, b)$ tel que calculé par l'algorithme correspond bien au plus grand motif commun entre $P(T(V))$ et $\pi'_i\pi'_{i+1}\cdots\pi'_j$, dont l'occurrence dans $\pi'_i\pi'_{i+1}\cdots\pi'_j$ n'utilise que des valeurs entre a et b, et qui est le plus petit dans l'ordre lexicographique.

Dans le cas où V est une feuille, le résultat est clair.

Dans le cas où V est un noeud interne ayant deux fils V_L et V_R, soient i, j, a et b tels que $1 \leq i \leq j \leq n'$ et $1 \leq a \leq b \leq n'$. Considérons le motif commun m de longueur maximale et le plus petit dans l'ordre lexicographique entre $P(T(V))$ et $\pi'_i\pi'_{i+1}\cdots\pi'_j$, dont l'occurrence dans $\pi'_i\pi'_{i+1}\cdots\pi'_j$ n'utilise que des valeurs entre a et b. Nécessairement, selon que la racine de $T(V)$ est un noeud positif ou négatif, on peut affirmer qu'il existe h et c tels que m se décompose

- soit en $m_l \oplus m_2$ avec m_1 (resp. m_2) un motif commun entre $P(T(V_L))$ et $\pi'_i\cdots\pi'_{h-1}$ utilisant des valeurs entre a et $c-1$ (resp. entre $P(T(V_R))$ et $\pi'_h\cdots\pi'_j$ utilisant des valeurs entre c et b) dans le cas d'une racine positive,
- soit en $m_1 \ominus m_2$ avec m_1 (resp. m_2) un motif commun entre $P(T(V_L))$ et $\pi'_i\cdots\pi'_{h-1}$ utilisant des valeurs entre c et b (resp. entre $P(T(V_R))$ et $\pi'_h\cdots\pi'_j$ utilisant des valeurs entre a et $c-1$) dans le cas d'une racine négative.

Remarquons que m_1 et m_2 peuvent être le motif vide.

On s'aperçoit facilement que si m_1 ou m_2 n'était pas de longueur maximale dans les intervalles d'indices et de valeurs donnés, alors m ne serait pas non plus de longueur maximale, et ceci contredirait la définition de m.

De même, si m_1 ou m_2 n'était pas le plus petit dans l'ordre lexicographique parmi les motifs de longueur maximale dans les intervalles d'indices et de valeurs donnés, on aurait une contradiction sur la définition de m.

On conclut donc que m_1 et m_2 sont de longueur maximale et les plus petits dans l'ordre lexicographique pour les intervalles d'indices et de valeurs données dans π'. Par hypothèse de récurrence, on a donc (pour le cas $m = m_1 \oplus m_2$, l'autre étant analogue) $m_1 = M(V_L, i, h-1, a, c-1)$ et $m_2 = M(V_R, h, j, c, b)$.

Le motif $M(V, i, j, a, b)$ tel que calculé par l'algorithme est donc "meilleur" que $m = m_1 \oplus m_2 = M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b)$ au sens d'être le plus long possible, et le plus petit dans l'ordre lexicographique parmi les plus longs. Or m est par définition maximal pour ces critères. On a donc démontré que $m = M(V, i, j, a, b)$.

Le cas où $m = m_1 \ominus m_2$ se traite de la même façon.

□

Reste à analyser la complexité de l'algorithme proposé. Il manipule un ensemble de tableaux dont la taille totale est $\mathcal{O}(nn'^4 min(n, n'))$, chaque case contenant un motif dont la taille est inférieure à $min(n, n')$. Pour remplir les tableaux $M(V, \_, \_, \_, \_)$ pour toutes les feuilles V, la complexité en temps est $\mathcal{O}(nn'^5)$. Et pour un noeud interne V, connaissant les tableaux associés aux fils de V, la complexité en temps pour remplir $M(V, \_, \_, \_, \_)$ est $\mathcal{O}(n'^6 min(n, n'))$. La complexité totale est donc $\mathcal{O}(nn'^6 min(n, n'))$. En fait, si on relâche la contrainte sur l'ordre lexicographique, on peut stocker dans les cases des tableaux seulement les tailles des motifs et des pointeurs vers les motifs des fils (i.e. construire à la volée l'arbre de séparation d'un plus grand motif commun), la complexité passant alors à $\mathcal{O}(nn'^6)$ en temps et $\mathcal{O}(nn'^4)$ en espace.

On peut donc énoncer le théorème suivant :

Théorème 4.3. *Le problème de recherche de plus grand motif commun à une permutation séparable et une permutation quelconque est dans P.*

Remarquons qu'on pourrait essayer d'utiliser une technique d'accélération comme dans l'algorithme d'Ibarra. Cette approche est peut-être possible, mais la transposition de l'algorithme d'Ibarra à la recherche de plus grand motif commun à deux permutations dont une séparable n'est pas aussi naturelle que pour l'algorithme de Bose,Buss et Lubiw. En effet, un point fondamental dans la correction de l'algorithme d'Ibarra est que $L(V, i, j, x)$ ne peut pas décroître lorsque i diminue. L'extension la plus naturelle de $L(V, i, j, x)$ serait de stocker dans $L(V, i, j, x)$ un plus grand motif commun à $P(T(V))$ et $\pi'_i\pi'_{i+1}\cdots\pi'_j$ dont l'occurrence dans $\pi'_i\pi'_{i+1}\cdots\pi'_j$ a son plus grand élément inférieur ou égal à x et son plus petit élément y le plus grand possible. Dans ce cas, si la taille du plus grand motif commun augmente, on peut, en faisant diminuer i, faire décroître la valeur du y concerné, ce qui annulerait la preuve de correction telle que la propose Ibarra. Cependant, je n'ai pas poursuivi dans cette voie, et la technique d'Ibarra est peut-être adaptable pour améliorer la complexité de l'algorithme décrit ici.

4.4 Distance entre permutations séparables

Comme on l'a vu dans le paragraphe précédent, on dispose d'un algorithme en temps polynomial pour la recherche du plus grand motif commun (qui est le plus petit dans l'ordre lexicographique) à deux permutations dont une est séparable. En particulier, étant données deux permutations séparables π_1 et π_2, on peut appliquer cet algorithme successivement à π_1 et π_2 puis à π_2 et π_1. Le fait d'avoir choisi un motif canonique parmi les motifs de longueur maximale (en l'occurrence le plus petit dans l'ordre lexicographique) fait que le motif m retourné par l'algorithme lors des deux appels sera le même.

Il est possible de tirer parti de cette particularité. En effet, si on retient dans l'algorithme où sont les occurrences du motif m dans π_1 et dans π_2, on obtient un "chemin de transformations" pour passer de π_1 à π_2 qui minimise le nombre de transformations. Pour formaliser cette idée, on introduit une distance d'édition sur les permutations.

Commençons par définir une opération d'insertion d'un élément dans une permutation. Soit $\pi = \pi_1\pi_2\ldots\pi_n$ une permutation de longueur n. Soit aussi un entier k entre 1 et $n+1$ que l'on veut insérer dans π, à une position p entre 1 et $n+1$. La permutation $\pi' = insert(\pi, k, p)$ de longueur $n+1$ est définie par :

$$\begin{array}{ll} \text{pour } 1 \leq j < p : & \left\{ \begin{array}{ll} \pi'_j = \pi_j & \text{si } \pi_j < k \\ \pi'_j = \pi_j + 1 & \text{si } \pi_j \geq k \end{array} \right. \\ \text{pour } j = p : \pi'_p = k & \\ \text{pour } p < j \leq n+1 : & \left\{ \begin{array}{ll} \pi'_j = \pi_{j-1} & \text{si } \pi_{j-1} < k \\ \pi'_j = \pi_{j-1} + 1 & \text{si } \pi_{j-1} \geq k \end{array} \right. \end{array}$$

Par exemple, $insert(52314, 2, 3) = 632415$.

L'opération inverse est la suppression d'un élément dans une permutation π de longueur $n+1$. Il suffit de donner la valeur de l'élément à supprimer. Contrairement au cas de l'insertion, il n'est pas nécessaire de fournir à la fois la valeur et la position.[1] On définit donc $\pi' = suppr(\pi, k)$ par

$$\begin{array}{ll} \text{pour } 1 \leq j < p : & \left\{ \begin{array}{ll} \pi'_j = \pi_j & \text{si } \pi_j < k \\ \pi'_j = \pi_j - 1 & \text{si } \pi_j > k \end{array} \right. \\ \text{pour } p \leq j \leq n : & \left\{ \begin{array}{ll} \pi'_j = \pi_{j+1} & \text{si } \pi_{j+1} < k \\ \pi'_j = \pi_{j+1} - 1 & \text{si } \pi_{j+1} > k \end{array} \right. \end{array}$$

où p est l'unique indice tel que $\pi_p = k$, i.e. $p = \pi_k^{-1}$.

Par exemple, $suppr(632415, 2) = 52314$.

On a évidement les égalités suivantes :

$$suppr(insert(\pi, k, p), k) = \pi \quad \text{et} \quad insert(suppr(\pi, k), k, \pi_k^{-1}) = \pi$$

On peut alors définir la distance d'édition $dist(\pi_1, \pi_2)$ entre deux permutations π_1 et π_2 comme le nombre minimal d'opérations (insertions et suppressions) qu'il faut effectuer à partir de π_1 pour obtenir π_2. $dist$ définit bien une distance : pour toutes permutations π_1, π_2 et π_3, on a

- $dist(\pi_1, \pi_2) = dist(\pi_2, \pi_1)$,
- $dist(\pi_1, \pi_2) = 0 \Leftrightarrow \pi_1 = \pi_2$,
- $dist(\pi_1, \pi_3) \leq dist(\pi_1, \pi_2) + dist(\pi_2, \pi_3)$.

En outre, pour toute suite d'insertions et de suppressions menant de π_1 à π_2 qui est la plus courte possible (i.e. contenant $dist(\pi_1, \pi_2)$ opérations), il est toujours possible d'effectuer d'abord toutes les suppressions puis toutes les insertions. En effet, si une insertion précède une suppression dans une telle suite d'opérations, l'élément supprimé n'est pas celui qui vient d'être inséré (sinon, la suite d'opérations ne serait pas la plus courte possible). On s'aperçoit alors facilement que l'on peut effectuer d'abord une suppression de l'élément à éliminer, puis une insertion du nouvel élément, pour aboutir au même résultat à l'issue des deux opérations. Ainsi, considérant une suite d'opérations la plus courte

[1] Remarquons qu'on aurait pu tout aussi bien choisir de donner seulement la position de l'élément à supprimer, mais on choisit (arbitrairement) de donner plutôt sa valeur.

possible, réordonnée en mettant d'abord les suppressions puis les insertions, on déduit grâce à la minimalité de cette suite que la permutation τ telle que

$$\pi_1 \xrightarrow{\text{suppressions}} \tau \xrightarrow{\text{insertions}} \pi_2$$

est un plus grand motif commun à π_1 et π_2.

De manière duale, l'algorithme de recherche de plus grand motif commun à deux permutations séparables peut être utilisé pour calculer la distance d'édition entre deux permutations séparables. Si π_1 et π_2 sont deux permutations dont le plus grand motif commun le plus petit dans l'ordre lexicographique (calculé par l'algorithme) est m, alors

$$dist(\pi_1, \pi_2) = |\pi_1| + |\pi_2| - 2|m|$$

où $|\pi|$ représente la taille de la permutation π.

On peut même remarquer que lors du déroulement de l'algorithme de recherche de plus grand motif commun m à π_1 et π_2, on peut aisément retenir aussi l'occurrence de m dans π_1. Et de même dans π_2, lorsque l'on appelle l'algorithme sur π_2 et π_1. Et comme on obtient bien le même motif m dans les deux appels de l'algorithme, on obtient deux suites de suppressions permettant de transformer π_1 en m d'une part, et π_2 en m d'autre part. Ces deux suites définissent une suite d'opérations qui réalise la distance d'édition entre π_1 et π_2. On peut donc non seulement calculer la distance d'édition entre permutations séparables grâce à l'algorithme de recherche de plus grand motif commun, mais on peut en outre trouver une réalisation de cette distance.

4.5 Ouverture vers les séries génératrices bivariées

Je conclus cette partie en faisant une remarque sur les possibilités qu'ouvre cette distance d'édition en termes de séries génératrices.

On a vu dans la partie sur l'énumération des permutations séparables qu'elles sont énumérées par les nombres de Schröder : il y a s_{n-1} permutations séparables de taille n. Ce qu'on aimerait faire est trouver une énumération des permutations séparables non seulement selon leur taille n, mais aussi selon la distance à la permutation identité de même taille $Id_n = 12\ldots n$. En fait, la distance entre deux permutations de même taille étant toujours paire, on considérera plutôt la demi-distance, qui correspond à la différence entre taille de la permutation et la taille de son plus grand motif commun avec Id_n. On notera la série génératrice associée par :

$$S(x, y) = \sum_{n \geq 0} \sum_{p \geq 0} s_{n,p} x^n y^p$$

$s_{n,p}$ représentant le nombre de permutations séparables de taille n à demi-distance p de Id_n.

Grâce à mes programmes Java, j'ai pu obtenir les premières valeurs $s_{n,p}$. Elles figurent dans le tableau donné en figure 9.

taille $n \rightarrow$ distance p à Id_n $\downarrow$	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1
1	0	1	4	9	16	25	36
2	0	0	1	11	45	125	281
3	0	0	0	1	27	178	703
4	0	0	0	0	1	64	634
5	0	0	0	0	0	1	150
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0
$\sum_p s_{n,p}$	1	2	6	22	90	394	1806

FIG. 9 – Tableau donnant le nombre de permutations de taille n à distance p de Id_n pour $n \leq 7$

On retrouve bien, comme le montre la dernière ligne du tableau, qu'il y a s_{n-1} permutations séparables de taille n.

On remarque dans le tableau ci-dessus que $s_{n,1} = (n-1)^2$ pour $1 \leq n \leq 7$. Ceci n'est en fait pas un hasard, comme le montre la propriété suivante :

Propriété 4.4. *Pour tout $n \geq 1$, il y a $(n-1)^2$ permutations séparables de taille n à demi-distance 1 de Id_n : $s_{n,1} = (n-1)^2$*

Démonstration. La preuve est en deux étapes. On commence par démontrer que toute permutation de taille n à demi-distance au plus 1 de Id_n est séparable. Ensuite, on exhibe une manière de générer sans doublons toutes les permutations à demi-distance 1 de Id_n à partir Id_{n-1}.

Montrons par récurrence sur $n \geq 1$ que toute permutation π de taille n à demi-distance 0 ou 1 de Id_n est séparable.
Pour $n = 1$, il n'y a que la permutation 1 à demi-distance au plus 1 de 1, et elle est évidemment séparable.
Soit π une permutation de taille $n+1$, à demi-distance 0 ou 1 de Id_{n+1}. Si π est à demi-distance 0 de Id_{n+1}, $\pi = Id_{n+1}$, et dans ce cas π est clairement séparable. Supposons donc que π est à demi-distance 1 de Id_{n+1}. π s'écrit $\widehat{\pi_1}(n+1)\widehat{\pi_2}$, où $\widehat{\pi_1}$ et $\widehat{\pi_2}$ sont des séquences de nombres distincts.
Si $\widehat{\pi_2}$ contient au moins un élément, l'élément $n+1$ doit être supprimé dans π puis réinséré à la fin pour obtenir Id_{n+1}. On a donc nécessairement $\pi = 1 \dots j(n+1)(j+1) \dots n$, et π est clairement séparable. Si au contraire $\widehat{\pi_2}$ est vide, on est dans le cas où $\widehat{\pi_1}$ est une permutation de taille n à distance 1 de Id_n. Par hypothèse de récurrence, $\widehat{\pi_1}$ est séparable, et on en déduit alors facilement que π est aussi séparable.

Je décris à présent une méthode permettant de générer toutes les permutations de taille n à demi-distance exactement 1 de Id_n, par insertion dans Id_{n-1}, et qui ne produit pas de doublons. On considère les n positions d'insertion : la position 1 avant le premier élément de Id_{n-1}, la position n après son der-

nier élément, et la position i entre son $(i-1)$-ème et son i-ème élément pour $2 \leq i \leq n-1$. L'insertion est celle de la fonction *insert* définie plus haut. Dans la position 1 on autorise l'insertion de toute valeur entre 2 et n, et dans toute position $i \geq 2$, on autorise l'insertion de tout entier k entre 1 et n, qui est différent de i et de $i-1$. Il est évident que l'insertion de la valeur i dans la position i produirait la permutation Id_n, qui ne vérifie pas la condition sur la demi-distance. Quant aux insertions des valeurs $i-1$ dans les position $i \geq 2$, elles produisent des permutations qui peuvent aussi être obtenues par insertion de la valeur i dans la position $i-1$. C'est donc pour éviter les doublons qu'on les exclut. On se convainc facilement que les permutations ainsi générées sont toutes celles à demi-distance 1 de Id_n, sans doublons. De manière synthétique :

$$\{\pi : \pi \text{ est à demi-distance } 1 \text{ de } Id_n\} =$$
$$\{insert(Id_{n-1}, k, 1) : k \neq 1\} \cup \bigcup_{i=2\ldots n} \{insert(Id_{n-1}, k, i) : k \neq i, k \neq i-1\}$$

On voit alors aisément que par cette méthode on construit $(n-1) + (n-1) \times (n-2) = (n-1)^2$ permutations distinctes de taille n à demi-distance 1 de Id_n. Elles sont toutes séparables d'après la première partie de la preuve. Ceci termine donc la démonstration de la propriété annoncée.

□

Remarquons enfin que les autres séquences qui apparaîssent dans le tableau ne sont pas répertoriées dans l'encyclopédie en ligne des séquences d'entiers [33], et il y a donc fort peu de chances qu'elles aient été déjà rencontrées dans un autre contexte.

À partir de la série génératrice $S(x, y)$, on voudrait pouvoir calculer la distance moyenne entre une permutation de taille n et Id_n, en utilisant la méthode des moments [19] comme dans [27]. Malheureusement, cette approche est infructueuse, car on n'a pas d'expression fonctionnelle (où les $s_{n,p}$ n'apparaîtraient pas) pour la série génératrice $S(x, y)$.

Au lieu de s'intéresser à cette distance moyenne, on va plutôt considérer la taille en moyenne de la plus longue sous-séquence croissante dans une permutation, qui n'est pas autre chose que la taille du plus grand motif commun entre une permutation et la permutation identité de même taille. Une astuce permet, à partir du théorème d'Erdös et Szekeres, de donner une borne inférieure sur la valeur moyenne de la taille de la plus grande sous-séquence croissante. Le théorème d'Erdös et Szekeres [18] énonce, dans sa version forte [35], que :

Théorème 4.5. *Soient p et q deux entiers, et $n = p \cdot q + 1$. Soit π une permutation de taille n. Alors π contient une sous-séquence croissante de taille $p+1$ ou une sous-séquence décroissante de taille $q+1$.*

Il a pour conséquence que [34] :

Propriété 4.6. *Dans l'ensemble des permutations de taille n, la longueur moyenne de la plus longue sous-séquence croissante est supérieure à $\sqrt{n-1}$.*

Avant de prouver cette propriété, j'ai besoin de quelques définitions.

Définition 4.7. *Soit $\pi = \pi_1\pi_2\ldots\pi_n$ une permutation. On définit son* renversé *π^r par $\pi^r = \pi_n\ldots\pi_2\pi_1$. Une classe de permutations R est dite* symétrique *si pour toute permutation $\pi \in R$, on a $\pi^r \in R$.*

Remarquons que l'ensemble des permutations de taille n est symétrique, pour tout entier n.

Je rappelle aussi que la concaténation négative de deux permutations π et π' de tailles respectives n et n' est définie par :

$$\pi \ominus \pi' = (\pi_1 + n') \cdots (\pi_n + n')\pi'_1 \cdots \pi'_{n'}$$

Démonstration. Pour toute permutation π, on note $is(\pi)$ (resp. $ds(\pi)$) la longueur de la plus grande sous-séquence croissante (resp. décroissante) dans π. On note $\mathbb{E}_n[f(\pi)]$ la moyenne de la fonction $f : S \to \mathbb{N}$ sur l'ensemble S_n des permutations de taille n. On a alors :

$$\begin{array}{rcl} \mathbb{E}_n[is(\pi)] & = & \frac{1}{|S_n|}\sum_{\pi\in S_n} is(\pi) \\ & = & \frac{1}{|S_n|}\sum_{\pi\in S_n} \frac{1}{2}(is(\pi) + is(\pi^r)) \\ & \geq & \frac{1}{|S_n|}\sum_{\pi\in S_n} \sqrt{is(\pi)\cdot is(\pi^r)} \\ & \geq & \frac{1}{|S_n|}\sum_{\pi\in S_n} \sqrt{is(\pi)\cdot ds(\pi)} \end{array}$$

Fixons $\pi \in S_n$, et montrons que $\sqrt{is(\pi)\cdot ds(\pi)} \geq \sqrt{n-1}$. Pour cela, posons $p = is(\pi)$ et $q = \lceil \frac{n-1}{p} \rceil$. Distinguons alors deux cas.
Si $\frac{n-1}{p} \in \mathbb{N}$, alors $n = p \cdot q + 1$, et par définition de p, l'utilisation du théorème d'Erdös-Szekeres donne directement que $ds(\pi) \geq q + 1 \geq q$, et ainsi l'on obtient que $\sqrt{is(\pi)\cdot ds(\pi)} \geq \sqrt{p\cdot q} \geq \sqrt{n-1}$.
Si au contraire $\frac{n-1}{p} \notin \mathbb{N}$, il existe un entier r, $0 < r < p$ tel que $n - 1 = p\cdot(q-1)+r$. On définit alors $n' = p\cdot q+1$ et $\pi' = \pi \ominus 12\ldots(p-r)$. π' est une permutation de longueur $n' \geq n$. On remarque alors que $is(\pi') = is(\pi) = p$ et que $ds(\pi') = ds(\pi) + 1$. Par le théorème d'Erdös-Szekeres, on obtient comme avant que $ds(\pi') \geq q + 1$, et donc $ds(\pi) \geq q$. On conclut alors que $\sqrt{is(\pi)\cdot ds(\pi)} \geq \sqrt{p\cdot q} = \sqrt{n'-1} \geq \sqrt{n-1}$.

On peut alors achever le calcul :

$$\begin{array}{rcl} \mathbb{E}_n[is(\pi)] & \geq & \frac{1}{|S_n|}\sum_{\pi\in S_n} \sqrt{is(\pi)\cdot ds(\pi)} \\ & \geq & \frac{1}{|S_n|}\sum_{\pi\in S_n} \sqrt{n-1} \\ & \geq & \sqrt{n-1} \end{array}$$

□

En fait, la preuve de ce résultat peut être utilisée telle quelle pour prouver la propriété suivante, un peu plus générale :

Propriété 4.8. *Soit R une classe de permutations* symétrique. *Dans l'ensemble des permutations de R de taille n, la longueur moyenne de la plus longue sous-séquence croissante est supérieure à $\sqrt{n-1}$.*

La classe $S(2413, 3142)$ des permutations séparables étant symétrique, on conclut que :

Propriété 4.9. *Dans l'ensemble des permutations séparables de taille n, la longueur moyenne de la plus longue sous-séquence croissante est supérieure à $\sqrt{n-1}$.*

5 Extension de l'algorithme de recherche de plus grand motif commun à un cadre plus général

Je reviens dans cette partie du rapport à des problèmes algorithmiques. Je propose une extension de l'algorithme polynomial de recherche de plus grand motif commun à une permutation séparable et une permutation générale, décrit dans la partie précédente, à une classe de permutations plus large.

Le point de départ de cette extension est une autre interprétation de l'arbre de séparation d'une permutation séparable. En effet, on peut pour tout graphe, et par extension pour toute permutation, construire son *arbre de décomposition*. L'arbre de décomposition d'une permutation séparable est très fortement lié à son arbre de séparation. On verra aussi que les permutations séparables sont caractérisées par des arbres de décomposition très particuliers : ne comportant que des *noeuds linéaires*. Il sera alors tout naturel d'étendre l'algorithme de recherche de plus grand motif commun à des permutations dont les arbres de décomposition sont un peu plus généraux que ceux des permutations séparables, ceux où les noeuds non-linéaires ont un degré borné. Mais avant cela, j'ai besoin de donner les bases de la théorie de la décomposition modulaire pour relier les arbres de séparation et les arbres de décomposition.

5.1 Éléments de décomposition modulaire des graphes, et application aux permutations

Pour rendre ce rapport le plus complet possible, je donne les définitions et propriétés de base de la décomposition modulaire des graphes. Cependant, je m'intéresserai essentiellement à la décomposition par intervalles communs d'une permutation, dont on peut prouver [28] qu'elle est "équivalente" à la décomposition modulaire d'un graphe de permutation. Il n'est pas nécessaire de connaître en détail la théorie de la décomposition modulaire pour comprendre comment elle s'adapte aux permutations, sous la forme de décomposition par intervalles communs.

Cas général des graphes

Définition 5.1. *Soit $G = (V, E)$ un graphe non orienté. Un* module *dans G est un sous-ensemble de sommets $V' \subset V$ qui jouent tous le même rôle vis-à-vis des sommets de $V \setminus V'$. Plus formellement, cela signifie que pour tout sommet $v \in V \setminus V'$, l'une des deux propositions suivantes est vraie :*
- *pour tout sommet v' de V', (v, v') est une arête de G*
- *il n'y a aucun sommet v' de V' tel que (v, v') soit une arête de G*

Un module est donc un ensemble de sommets qui, vu de l'extérieur du module, se comporte comme un seul sommet. Dans la figure 10, les modules sont dessinés par des "patates", une arête allant du module V' vers un sommet v si et seulement si (v, v') est une arête du graphe pour tout $v' \in V'$.

Dans n'importe quel graphe $G = (V, E)$, certains modules existent toujours : on les appelle *modules triviaux*. Il s'agit de $\emptyset$, V, et tous les singletons $\{v\}$ pour $v \in V$. Les graphes dont les seuls modules sont les modules triviaux sont appelés *graphes premiers*.

Certains modules dans un graphe revêtent une importance particulière :

Définition 5.2. *On dit qu'un module dans un graphe est un* module fort *lorsqu'il ne chevauche*[2] *aucun autre module.*

Remarquons que tout module trivial est un module fort.

Considérons à présent deux modules forts d'un graphe $G = (V, E)$. Comme ils ne se chevauchent pas, on est forcément dans un des trois cas suivants : $A \subseteq B$ ou $B \subseteq A$ ou $A \cap B = \emptyset$. Les modules forts peuvent donc être ordonnés entre eux par l'ordre d'inclusion, et l'ordre d'inclusion sur les modules forts est un ordre arborescent : on peut construire un arbre dont les noeuds sont les modules forts (hormis le module vide), avec comme feuilles les singletons $\{v\}$ pour $v \in V$, comme racine le module V, et tel qu'un module fort V' ait pour descendants tous les modules forts V'' tels que $V'' \subset V'$. Cet arbre est *l'arbre de décomposition modulaire* de G. Remarquons que cet arbre n'est pas un arbre plan, c'est-à-dire qu'il n'y a pas d'ordre entre les fils.

Les noeuds internes de l'arbre de décomposition modulaire, qui correspondent à des modules forts, peuvent être classés dans deux catégories : les noeuds *complets* et les noeuds *premiers*. Ces notions sont définies en termes de *quotient de graphes* dans [28]. La propriété suivante [28] en donne une caractérisation :

Propriété 5.3. *Tout noeud interne V de l'arbre de décomposition modulaire d'un graphe G est :*
- *soit* complet *: n'importe quelle partie des fils de V donne, lorsqu'on en fait l'union, un module,*
- *soit* premier *: l'union de plusieurs fils de V n'est un module que dans le cas où l'on prend* tous *les fils de V.*

Cette propriété permet donc d'étiqueter les noeuds de tout arbre de décomposition modulaire, avec les étiquettes "complet" ou "premier".

La figure 10 illustre les notions introduites.

Il est à remarquer aussi que le calcul de l'arbre de décomposition modulaire d'un graphe peut être fait en temps polynomial, et il existe même des algorithmes en temps linéaire en la taille du graphe qui calculent son arbre de décomposition modulaire [6, 28].

Cas particulier des permutations Considérons maintenant une permutation π. On pourrait définir l'arbre de décomposition de π en intervalles communs comme étant l'arbre de décomposition du graphe de la permutation π, rendu plan en imposant que les feuilles apparaissent dans l'ordre de π, et modulo un petit changement dans l'étiquetage. Cependant, je trouve plus claire la construction directe donnée dans [13], et je la décris ci-dessous. L'équivalence des deux constructions est démontrée dans [28].

[2] La définition du chevauchement suit l'intuition qu'on a de cette notion : deux ensembles de sommets A et B se chevauchent lorsque $A \setminus B \neq \emptyset$, $B \setminus A \neq \emptyset$ et $A \cap B \neq \emptyset$

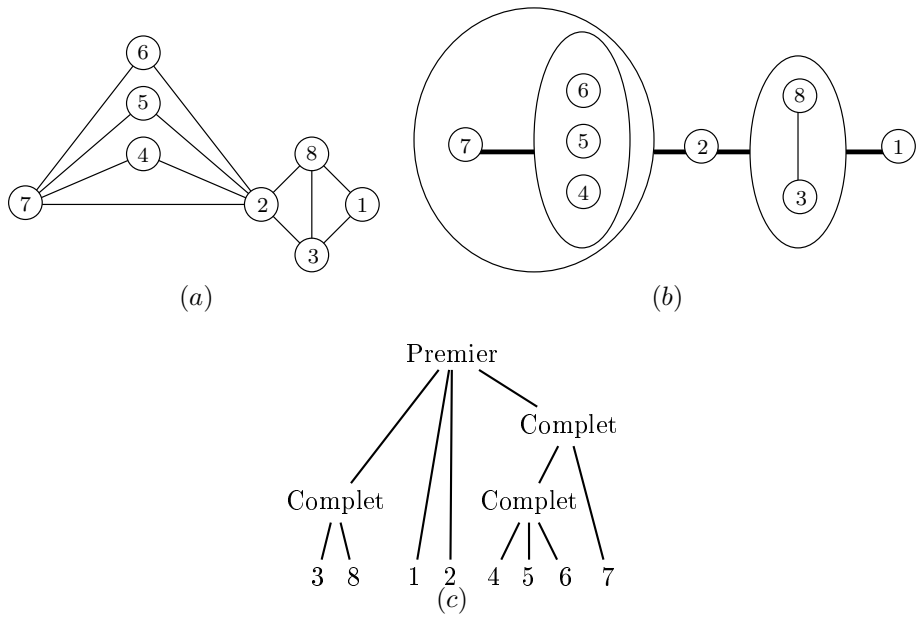


FIG. 10 – Un graphe (a), sa représentation par modules (b), et son arbre de décomposition modulaire (c)

La *décomposition en intervalles communs* d'une permutation π de taille n (sous-entendu, intervalles communs avec Id_n) se définit comme suit. On considère d'abord la famille de tous les intervalles communs de π, i.e. les sous-permutations $I = \pi_j \pi_{j+1} \dots \pi_k$ (les indices devant être consécutifs) telles que $\{\pi_j, \pi_{j+1}, \dots, \pi_k\}$ soit un intervalle de $\mathbb{N}$ (au sens usuel du terme). On notera alors $val(I) = \{\pi_j, \pi_{j+1}, \dots, \pi_k\}$: $val(I)$ est un intervalle de $\mathbb{N}$. On notera aussi $pos(I) = \{j, j+1, \dots, k\}$, l'ensemble des positions de π occupées par l'intervalle commun I. $pos(I)$ est aussi un intervalle de $\mathbb{N}$. Dans la suite, je note souvent I pour $val(I)$ ou pour $pos(I)$ lorsqu'il n'y a pas de confusion possible.

Dans la famille des intervalles communs, on définit les intervalles communs forts (abrégé dans la suite en intervalles forts) comme étant ceux qui ne chevauchent[3] aucun autre intervalle commun. Par exemple, $\pi, 1, 2, \dots, n$ sont des intervalles forts. L'ensemble des intervalles forts peut être ordonné pour l'ordre d'inclusion. Deux intervalles forts ne pouvant se chevaucher, l'ordre d'inclusion des intervalles forts entre eux est arborescent : il est donné par un arbre dont les feuilles sont π_1, π_2, ..., π_n (de gauche à droite dans cet ordre), dont la racine est π, et dont chaque noeud interne est obtenu par réunion de ses noeuds fils[4].

Pour illustrer ces définitions, la figure ci-dessous représente la permutation

[3] Comme dans le cas des graphes, on dit que deux intervalles communs I et J se chevauchent lorsque $pos(I) \setminus pos(J) \neq \emptyset$, $pos(J) \setminus pos(I) \neq \emptyset$ et $pos(I) \cap pos(J) \neq \emptyset$

[4] J'utilise le terme réunion ici bien qu'il ne s'agisse pas d'une réunion ensembliste. J'entends que, si le noeud interne est I, avec pour fils $I_1, \dots, I_k$, alors I est la concaténation $I_1 \cdot I_2 \dots \cdot I_k$. En particulier, $val(I) = \cup_{i=1}^{k} val(I_i)$ et $pos(I) = \cup_{i=1}^{k} pos(I_i)$.

53412 et deux intervalles communs $I_1 = 534$ et $I_2 = 3412$ se chevauchant. On a $val(I_1) = \{3, 4, 5\}$, $pos(I_1) = \{1, 2, 3\}$, $val(I_2) = \{1, 2, 3, 4\}$ et $pos(I_2) = \{2, 3, 4, 5\}$.

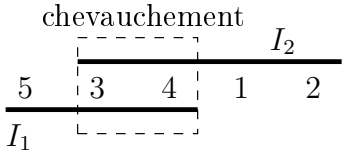


FIG. 11 – Deux intervalles communs se chevauchant de la permutation 53412

On remarque alors qu'il existe deux types de noeuds internes dans l'arbre des intervalles forts. Pour certains noeuds V ayant k fils $V_1, V_2, \dots V_k$, aucune réunion de i fils pour n'importe quel i entre 2 et $k-1$ n'est un intervalle commun. Ces noeuds sont appelés noeuds *premiers* et sont étiquetés par P.

Les autres noeuds sont tels que l'ordre $\prec$ sur leurs fils $V_1, V_2, \dots V_k$ défini par $V_i \prec V_j$ si et seulement si V_i est à gauche de V_j est un ordre linéaire vérifiant la propriété que les unions de fils de V qui sont des intervalles communs sont exactement les réunions de i fils consécutifs pour l'ordre $\prec$, pour $1 \leq i \leq k$: $\cup_{j=d+1}^{d+i} V_j$. Ces noeuds sont appelés noeuds *linéaires* et sont étiquetés par L.

L'arbre étiqueté obtenu est appelé *arbre de décomposition* de π. L'ordre des feuilles de gauche à droite étant imposé par π, cet arbre est un arbre plan, au contraire de l'arbre de décomposition d'un graphe.

Remarquons que les noeuds d'arité 2 satisfont à la fois la définition de noeud premier et celle de noeud linéaire. On choisit de les étiqueter par L. Attention, ce choix n'est pas arbitraire et il sera expliqué un peu plus loin. D'autre part, toutes les propriétés de l'arbre de décomposition que j'ai énoncées, comme le fait qu'un noeud soit toujours de type premier ou de type linéaire, sont démontrées dans [13]. Il est aussi important de savoir qu'il existe des algorithmes [6, 13] qui calculent l'arbre de décomposition d'une permutation de taille n en temps $\mathcal{O}(n)$.

Pour clarifier les notions d'intervalles forts et d'arbre de décomposition, je traite un exemple en figure 12 et 13.

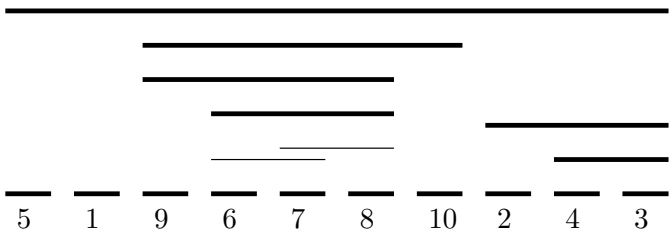


FIG. 12 – Décomposition en intervalles communs de $\pi = 519678(10)243$.
Les intervalles communs sont représentés par des traits horizontaux, les traits gras correspondant aux intervalles forts.

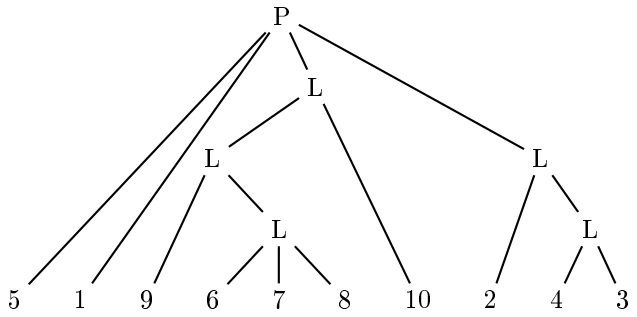


FIG. 13 – Arbre de décomposition de $\pi = 519678(10)243$

5.2 Une définition équivalente des permutations séparables et de l'arbre de séparation

Après avoir défini l'arbre de décomposition d'une permutation quelconque, il est tout naturel dans le contexte des permutations séparables, de se demander si leurs arbres de décomposition ont des propriétés caractéristiques. Je réponds à cette question par l'affirmative, en donnant même une caractérisation des permutations séparables à travers leur arbre de séparation.

Mais d'abord, la propriété suivante montre qu'il existe un lien très fort entre arbre n-aire de séparation et arbre de décomposition d'une permutation séparable.

Propriété 5.4. *Soit π une permutation séparable de taille n, T_s son arbre n-aire de séparation, et T_d son arbre de décomposition. Alors, si l'on ne tient pas compte des étiquettes des noeuds, $T_d = T_s$.*

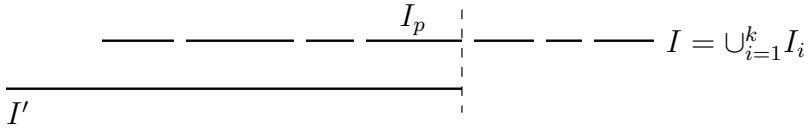
Démonstration. Remarquons d'abord que la relation père-fils est définie de la même manière dans T_s et dans T_d : les fils d'un noeud interne V représentent des intervalles communs inclus dans l'intervalle commun représenté par V. Pour conclure que $T_d = T_s$, il suffit donc de montrer que ces deux arbres ont même ensemble de sommets. En effet, l'ordre des feuilles étant le même dans les deux arbres, on en déduit alors que T_s et T_d sont identiques en tant qu'arbres plans (hormis l'étiquetage des noeuds). Rappelons que les noeuds de T_d sont les intervalles forts de π. Il suffit donc de démontrer que :

1. tout noeud de l'arbre de séparation représente un intervalle fort,
2. toute sous-permutation $\pi_a\pi_{a+1}\dots\pi_b$ (les positions étant consécutives) qui n'est pas un noeud de l'arbre de séparation n'est pas un intervalle fort.

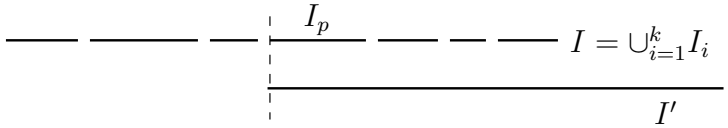
1. On procède par récurrence structurelle sur les noeuds de T_s.
   - Pour une feuille V de T_s, V représente un certain entier i entre 1 et n, et i est clairement un intervalle fort.
   - Pour un noeud interne V de T_s ayant pour fils $V_1, V_2, \dots, V_k$, on sait par hypothèse de récurrence que chaque V_i correspond à un intervalle fort I_i de π. Par définition de l'arbre de séparation, le noeud V représente un intervalle commun de π, que l'on note I. I est la réunion des I_i pour $1 \leq i \leq k$. Raisonnons par l'absurde et supposons que I ne soit pas un

intervalle fort. Alors I chevauche un autre intervalle commun de π : I'. Comme tous les I_i sont forts, ils ne chevauchent pas I', et on est donc dans l'une des deux situations suivantes :

– il existe $p \geq 1$ tel que pour tout $i \leq p$, $I_i \subseteq I'$ et pour tout $i > p$, $I_i \cap I' = \emptyset$

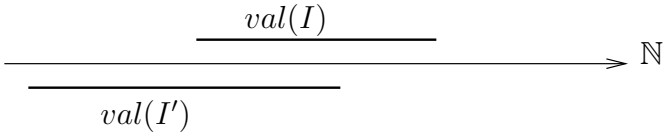


– il existe $p \leq k$ tel que pour tout $i \geq p$, $I_i \subseteq I'$ et pour tout $i < p$, $I_i \cap I' = \emptyset$



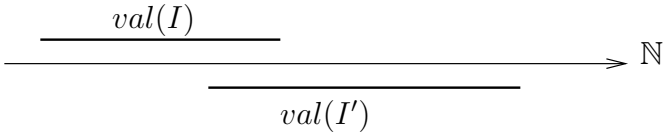
Dans ce qui suit je traite seulement le premier cas, le deuxième étant analogue. Plaçons nous donc dans le premier cas, et distinguons deux sous-cas.

– Si V est un noeud positif, alors $Min(val(I)) \in val(I_1)$, donc $val(I)$ et $val(I')$ sont des intervalles de $\mathbb{N}$ qui se chevauchent et tels que $Min(val(I)) \in val(I) \cap val(I')$. En repérant $val(I)$ et $val(I')$ sur une droite représentant $\mathbb{N}$, on est donc dans la situation suivante :



Donc $I' \setminus I$ est un intervalle commun de π (car c'est à la fois un intervalle en positions et en valeurs). En outre, les valeurs de $val(I' \setminus I)$ sont toutes strictement plus petites que $Min(val(I))$. V étant un noeud positif, cela implique que dans la construction de l'arbre de séparation T_s de π, $V_1, \ldots V_k$ auraient dû avoir au moins un frère en plus à gauche de V_1. D'où une contradiction.

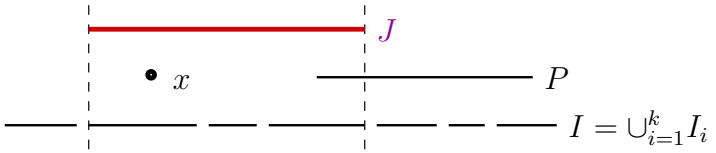
– Si V est un noeud négatif, alors $Max(val(I)) \in val(I_1)$, donc $val(I)$ et $val(I')$ sont des intervalles de $\mathbb{N}$ qui se chevauchent et tels que $Max(val(I)) \in val(I) \cap val(I')$. En repérant $val(I)$ et $val(I')$ sur une droite représentant $\mathbb{N}$, on est donc dans la situation suivante :



Donc $I' \setminus I$ est un intervalle commun de π (car c'est à la fois un intervalle en positions et en valeurs). En outre, les valeurs de $val(I' \setminus I)$ sont toutes strictement plus grandes que $Max(val(I))$. V étant un noeud négatif, cela implique que dans la construction de l'arbre de séparation T_s de π, $V_1, \ldots V_k$ auraient dû avoir au moins un frère en plus à gauche de V_1. D'où une contradiction.

Dans tous les cas, on aboutit à une contradiction, et on conclut ainsi que I est un intervalle fort.

2. Soit P une suite de positions consécutives dans π ($P = \pi_a \pi_{a+1} \dots \pi_b$) qui ne correspond à aucun noeud de l'arbre de séparation. Si P n'est pas un intervalle commun, P n'est pas un intervalle fort. Si au contraire P est un intervalle commun, considérons l'intervalle commun I associé à un noeud de l'arbre de séparation tel que $P \subseteq I$, avec I choisi le moins large possible. Remarquons que l'organisation en arbre des intervalles communs correspondant à des noeuds de T_s impose l'unicité de l'intervalle I défini. Par hypothèse sur P, $P \neq I$, donc il existe x tel que $x \in I$ et $x \notin P$. P n'étant pas vide, I contient donc au moins 2 éléments, donc correspond à un noeud interne de T_s. Notons $I_1, \dots I_k$ les fils de I. Par définition de l'arbre de séparation, on sait que pour $1 \leq d_1 \leq d_2 \leq k$, $\cup_{i=d_1}^{d_2} I_i$ est un intervalle commun de π. Considérons alors l'intervalle commun $J = \cup_{i=d_1}^{d_2} I_i$ avec d_1 le plus grand possible et d_2 le plus petit possible tel que $x \in J$ et $P \cap J \neq \emptyset$.



Comme $x \notin P$, on a $J \setminus P \neq \emptyset$. Enfin supposons que $P \subseteq J$. Par choix de J, cela implique qu'il existe i entre d_1 et d_2 tel que $P \subseteq I_i$, ce qui contredit le choix de I comme le moins large des intervalles contenant P parmi les noeuds de T_s. On déduit ainsi que $P \setminus J \neq \emptyset$. Avec $J \setminus P \neq \emptyset$ et $P \cap J \neq \emptyset$, on conclut que J chevauche P, et donc que P n'est pas un intervalle fort.

□

Ayant établi cette identité entre arbres de séparation et arbres de décomposition, on remarque alors qu'un noeud V de l'arbre de séparation (que ce soit un noeud positif ou un noeud négatif) possède la propriété que les unions de fils de V qui sont des intervalles communs sont exactement les réunions de fils de V consécutifs dans l'ordre de gauche à droite. Cette propriété est celle définissant les noeuds linéaires. Avec la propriété précédente et cette remarque, on déduit que :

Propriété 5.5. *Dans l'arbre de décomposition d'une permutation séparable, tous les noeuds sont linéaires.*

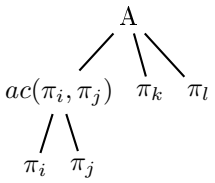
C'est pour pouvoir énoncer cette propriété de manière élégante qu'il faut choisir d'étiqueter par L les noeuds d'arité 2, sinon on aurait un énoncé du genre : tous les noeuds sont linéaires sauf peut-être ceux d'arité 2. En outre, la propriété d'être un noeud linéaire va être plus intéressante pour des manipulations algorithmiques que la propriété d'être un noeud premier, et mieux vaut donc utiliser au maximum la structure linéaire des noeuds d'arité 2.

En fait, on peut en dire plus sur les arbres de décomposition des permutations séparables, avec le théorème suivant qui est une nouvelle caractérisation des permutations de cette classe :

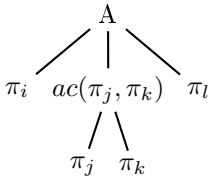
Théorème 5.6. *Les permutations séparables sont exactement celles dont l'arbre de décomposition ne contient que des noeuds linéaires.*

Démonstration. On a déjà démontré que les arbres de décomposition des permutations séparables ne contiennent que des noeuds linéaires. Il ne reste que la réciproque : dans l'arbre de décomposition d'une permutation qui n'est pas séparable, il y a au moins un noeud premier. Évidemment, je considère que les noeuds d'arité 2 ne sont pas des noeuds premiers.
Soit donc π une permutation qui n'est pas séparable, et soit T son arbre de décomposition. Pour un ensemble S de feuilles de T, je noterai $ac(S)$ le plus petit ancêtre commun à toutes les feuilles de S dans T. Comme π n'est pas séparable, il existe $i < j < k < l$ tels que $\pi_i\pi_j\pi_k\pi_l$ soit une occurrence de 3142 (ou de 2413, ce cas se traitant de la même manière). Démontrons d'abord de $ac(\pi_i, \pi_j) = ac(\pi_j, \pi_k) = ac(\pi_k, \pi_l) = ac(\pi_i, \pi_j, \pi_k, \pi_l)$. Notons $A = ac(\pi_i, \pi_j, \pi_k, \pi_l)$.

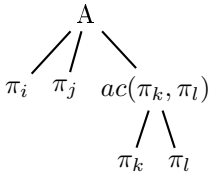
- $ac(\pi_i, \pi_j)$ couvre un intervalle commun I qui contient π_i et π_j. Or $\pi_j < \pi_l < \pi_i$, donc $\pi_l \in I$. Et π_k étant positionné entre π_j et π_l, on a aussi $\pi_k \in I$. Donc $ac(\pi_i, \pi_j)$ est un ancêtre commun de π_i, de π_j, de π_k et de π_l, et ça ne peut être que le plus petit. Donc $ac(\pi_i, \pi_j) = A$.

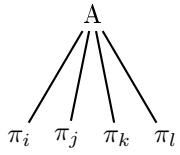


- $ac(\pi_j, \pi_k)$ couvre un intervalle commun I qui contient π_j et π_k. Or $\pi_j < \pi_l < \pi_i < \pi_k$, donc $\pi_l \in I$ et $\pi_i \in I$. Donc $ac(\pi_j, \pi_k)$ est un ancêtre commun de π_i, de π_j, de π_k et de π_l, et ça ne peut être que le plus petit. Donc $ac(\pi_j, \pi_k) = A$.



- $ac(\pi_k, \pi_l)$ couvre un intervalle commun I qui contient π_k et π_l. Or $\pi_l < \pi_i < \pi_k$, donc $\pi_i \in I$. Et π_j étant positionné entre π_i et π_k, on a aussi $\pi_j \in I$. Donc $ac(\pi_k, \pi_l)$ est un ancêtre commun de π_i, de π_j, de π_k et de π_l, et ça ne peut être que le plus petit. Donc $ac(\pi_k, \pi_l) = A$.





Considérons alors le noeud A, qui a au moins 4 fils, de gauche à droite I_i, I_j, I_k et I_l, chaque I_p étant un ancêtre de π_p. $\pi_i\pi_j\pi_k\pi_l$ étant une occurrence de 3142, la réunion des fils de A entre I_i et I_j (par exemple) n'est pas un intervalle. Donc A n'est pas un noeud linéaire. C'est donc que A est un noeud premier.

□

Cette caractérisation des permutations séparables permet de les décrire dans un contexte différent de celui des permutations à motifs exclus, et grâce à un objet préexistant, que l'on peut définir pour n'importe quelle permutation : l'arbre de décomposition. Cette nouvelle description, ne faisant pas appel à une construction aussi spécifique que l'arbre de séparation, n'est pas nécessairement préférable, mais elle est indéniablement un argument qui vient étayer les motivations pour l'étude des permutations séparables.

5.3 Décoration des arbres de décomposition des permutations

L'idée qui vient ensuite est d'utiliser l'arbre de décomposition au lieu de l'arbre de séparation pour rechercher algorithmiquement des motifs communs. Cependant, l'étiquetage de l'arbre de décomposition n'est pas assez précis pour cela, et il faut le *décorer*, i.e. lui superposer un étiquetage supplémentaire.

On se rend compte assez facilement que, pour tout noeud linéaire, les intervalles de ses fils sont classés par valeurs croissantes ou par valeurs décroissantes, et qu'il n'y a pas d'autre possibilité. Les noeuds linéaires vont donc se répartir en deux catégories : les noeuds positifs (d'étiquette $+$) et les noeuds négatifs (d'étiquette $-$). Pour une permutation séparable, en décorant ainsi son arbre de décomposition, on retrouve son arbre n-aire de séparation. L'étiquetage des noeuds premiers est assez différent, et demande de s'intéresser à une classe particulière de permutations : les *permutations simples*.

Les permutations simples sont définies dans la littérature [2, 3, 11, 12] comme celles ayant pour seuls intervalles communs avec l'identité de même taille (disons n) les intervalles triviaux $\{1\}, \ldots, \{n\}$ et $\{1, \ldots, n\}$. En d'autres termes, ce sont celles dont l'arbre de décomposition est composé d'un unique noeud interne, premier, auquel sont accrochées toutes les feuilles $\pi_1, \ldots, \pi_n$. Des études ont déjà été menées et d'autres sont en cours sur les permutations simples. On connaît en particulier un équivalent asymptotique de leur énumération [3, 24] : le nombre de permutations simples de taille n est équivalent à $\frac{n!}{e^2}$ lorsque n tend vers $+\infty$, c'est-à-dire qu'une grande proportion des permutations sont simples ! Ce résultat contraste avec les permutations séparables, comme d'ailleurs avec toutes les classes de permutations à motifs exclus.

Les permutations simples vont donc me servir à étiqueter les noeuds premiers. Les fils d'un noeud premier étant des intervalles, j'étiquette un noeud premier V à k fils par une permutation simple σ de taille k qui donne l'ordre relatif des fils de V entre eux : $\sigma_i < \sigma_j$ si et seulement si l'intervalle du i-ème fils de V contient des valeurs toutes inférieures à celles de l'intervalle du j-ème fils de V. Par exemple, sur la figure 13, la permutation simple qui étiquette la racine est 3142.

Pour pouvoir manipuler cet arbre de décomposition décoré, il me reste à analyser la complexité de son calcul. Rappelons que l'arbre de décomposition d'une permutation peut être calculé en temps linéaire [13, 6]. Une fois calculé l'arbre de décomposition, dont les noeuds sont en nombre $\mathcal{O}(n)$, on peut décorer ses noeuds. Il suffit d'une opération de comparaison pour décorer un noeud linéaire : on compare une valeur v_1 de l'intervalle du premier fils à une valeur v_2 de l'intervalle du dernier fils, et on décore le noeud par $+$ si $v_1 < v_2$, par $-$ si $v_1 > v_2$. Pour un noeud premier d'arité d, il suffit de choisir une valeur v_i pour chaque intervalle fils du noeud (v_i pour le i-ème intervalle le plus à gauche), et de "normaliser" la séquence $v_1 \ldots v_d$, i.e. de calculer la permutation σ de longueur d telle que $\sigma_i < \sigma_j$ si et seulement si $v_i < v_j$. Il suffit pour cela de savoir trier les $(v_i)_{1 \leq i \leq d}$, ce que l'on peut faire en temps $\mathcal{O}(d \log d)$. Au total, on peut donc décorer un arbre de décomposition en temps $\mathcal{O}(d \log d \cdot n)$, où d est une borne

sur l'arité des noeuds premiers, donc en particulier en temps $\mathcal{O}(n^2 \log n)$.

Ce qui m'intéresse ici est que le calcul de l'arbre de décomposition décoré d'une permutation puisse être fait en temps *polynomial*.

5.4 Adaptation de l'algorithme de recherche de plus grand motif commun dans un cadre plus général, avec l'outil des arbres de décomposition

L'algorithme de recherche de plus grand motif commun à une permutation séparable et une permutation quelconque utilisait la structure d'arbre de séparation *binaire* des permutations séparables. Or ce sont les arbres de séparation *n-aires* des permutations séparables qui sont égaux à leurs arbres de décomposition décorés. Le premier pas vers une nouvelle version de cet algorithme, qui utiliserait les arbres de décomposition, est donc de le modifier pour qu'il utilise plutôt les arbres n-aires de séparation.

En fait, la manière la plus simple de voir que l'on peut utiliser l'arbre n-aire plutôt que l'arbre binaire de séparation, est de remarquer qu'on peut "développer" un arbre n-aire de séparation pour en faire un arbre binaire de séparation. On transforme tout noeud positif (resp. négatif) ayant strictement plus de deux fils $V_1, V_2, \ldots, V_k$ en un noeud positif (resp. négatif) ayant deux fils V_1 et V', V' étant un nouveau noeud positif (resp. négatif) ayant pour fils $V_2, \ldots, V_k$. Et on poursuit récursivement la transformation sur V'. Ce développement est en quelque sorte une transformation "inverse" de la fusion que j'avais décrite pour construire les arbres de séparation n-aires. Évidemment, comme plusieurs arbres binaires avaient pour image le même arbre n-aire par l'opération de fusion, cette opération n'est pas réversible. Mais en revanche si on développe un arbre n-aire T puis qu'on fusionne l'arbre obtenu, on retrouve T.

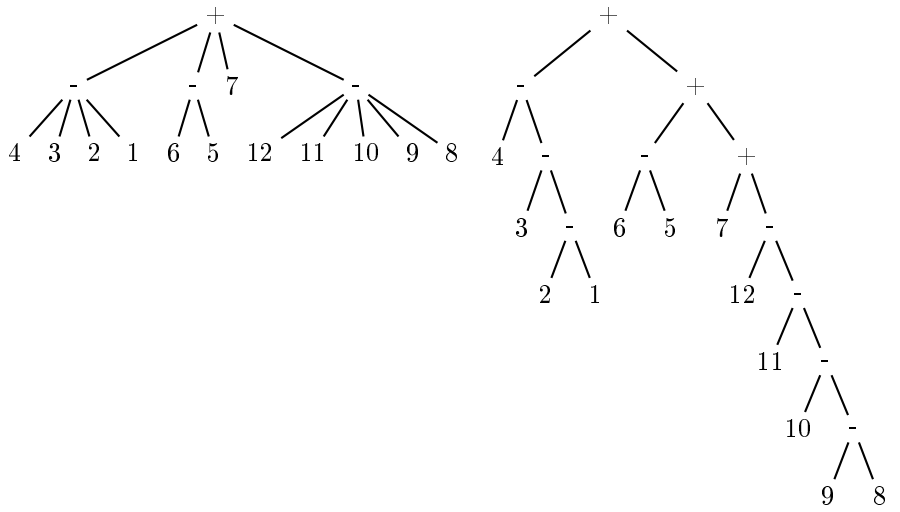


FIG. 14 – Exemple de développement d'un arbre n-aire de séparation

On vérifie bien que le développement d'un arbre n-aire de séparation T est

une opération qui peut être faite en temps linéaire. En effet, dans l'arbre binaire T' obtenu à la fin du développement, il y a $n-1$ noeuds internes (n étant le nombre de feuilles de T, i.e. encore la taille de la permutation dont T est un arbre de séparation), puisque cet arbre est binaire. On a donc effectué moins de $n-1$ développements de noeuds pour obtenir T', chacun de ces développements élémentaires demandant un temps constant.

L'opération de développement des arbres n-aires de séparation peut être étendue aux arbres de décomposition. Considérons un arbre de décomposition décoré T. Le développement de T est l'arbre obtenu en laissant inchangés les noeuds premiers, et en appliquant aux noeuds linéaires (positifs ou négatifs) la même transformation que dans les arbres de séparation. On obtient alors un arbre (que j'appellerai dans la suite arbre de décomposition par souci de brièveté) où les noeuds premiers sont d'arité quelconque, mais où les noeuds linéaires sont tous d'arité 2. Là encore, le développement d'un arbre de décomposition peut être fait en temps linéaire en le nombre de feuilles de cet arbre.

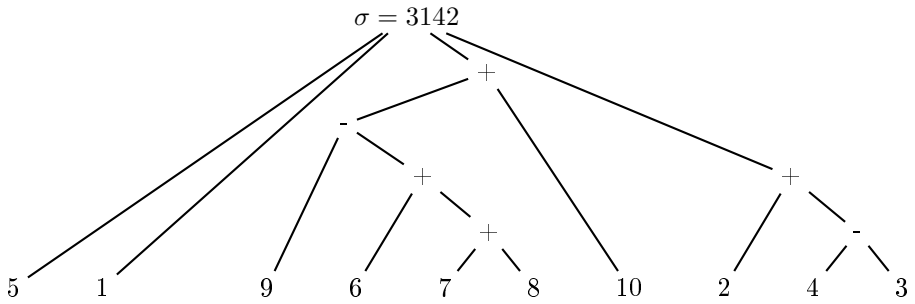


FIG. 15 – Développement de l'arbre de décomposition de la permutation $\pi = 519678(10)243$, arbre donné en figure 13

On voudrait à présent travailler sur les arbres de décomposition (décorés et développés), et donner un algorithme sur le modèle de celui de la section 4 pour la recherche de plus grand motif commun à deux permutations. On peut *a priori* choisir deux permutations quelconques, l'arbre de décomposition étant défini pour n'importe quelle permutation. Bien entendu, on n'attend pas que cet algorithme soit polynomial puisque le problème de recherche d'un plus grand motif commun à deux permutations quelconques est NP-dur. Cependant, on espère ainsi exhiber des classes de permutations pour lesquelles l'algorithme sera polynomial, et qui soient plus grandes que celle des permutations séparables.

Avant de décrire l'algorithme qui généralise celui de la section 4, je rappelle la définition de la concaténation positive $\oplus$ et de la concaténation négative $\ominus$: pour deux motifs p et p' de tailles respectives k et k', on définit

$$p \oplus p' = p_1 \cdots p_k (p'_1 + k) \cdots (p'_{k'} + k) \text{ et}$$

$$p \ominus p' = (p_1 + k') \cdots (p_k + k') p'_1 \cdots p'_{k'}$$

Je définis aussi une nouvelle concaténation pour chaque permutation simple σ : étant donnés σ une permutation simple de taille n et n motifs $p_1, \ldots, p_n$ de

tailles respectives $k_1, \ldots, k_n$, la σ-concaténation des $(p_i)_{1 \leq i \leq n}$ est :

$$\sigma(p_1, \ldots, p_n) = \text{shift}(p_1, \sigma_1) \ldots \text{shift}(p_n, \sigma_n) \text{ où}$$

$$\text{shift}(p_i, \sigma_i) = \text{shift}(p_i, \sigma_i)(1) \ldots \text{shift}(p_i, \sigma_i)(k_i) \text{ et}$$

$$\text{shift}(p_i, \sigma_i)(x) = (p_i(x) + k_{\sigma_1^{-1}} + \ldots + k_{\sigma_{i-1}^{-1}}) \text{ pour tout } x \text{ entre 1 et } k_i$$

Par exemple, $25314(21, 312, 4321, 12, 231) = 4\ 3\ 14\ 12\ 13\ 8\ 7\ 6\ 5\ 1\ 2\ 10\ 11\ 9$.

L'algorithme fonctionne comme avant par programmation dynamique. Il prend en entrée deux permutations π et π', de tailles respectives n et n', et commence par calculer l'arbre de décomposition T de π, étape qui peut être faite en temps polynomial en n, comme expliqué avant. De la même façon que dans la section 4, on note $T(V)$ le sous-arbre de T de racine V pour un noeud V de T, et $P(T(V))$ la sous-permutation de π dont $T(V)$ est un arbre de décomposition.

L'algorithme remplit ensuite un tableau à quatre dimensions pour chaque noeud V de T : $M(V, \_, \_, \_, \_)$, de telle manière qu'à la fin, $M(V, i, j, a, b)$ contienne un plus grand motif commun à $P(T(V))$ et $\pi'_i \pi'_{i+1} \cdots \pi'_j$, dont l'occurrence dans $\pi'_i \pi'_{i+1} \cdots \pi'_j$ n'utilise que des valeurs entre a et b, pour $1 \leq i \leq j \leq n'$ et $1 \leq a \leq b \leq n'$. Si $i > j$ ou $a > b$, $M(V, i, j, a, b)$ est le motif vide. Le remplissage des tableaux est donc par programmation dynamique :

- Pour un noeud V de T qui est une feuille, $M(V, i, j, a, b)$ est le motif 1 s'il existe $h \in \{i, i+1, \ldots, j\}$ tel que $a \leq \pi'_h \leq b$, le motif vide sinon.
- Pour un noeud interne linéaire positif V de T, ayant pour fils V_L et V_R, $M(V, i, j, a, b)$ est un motif de longueur maximale dans l'ensemble $\{M(V_L, i, j, a, b), M(V_R, i, j, a, b)\}$ $\cup \{M(V_L, i, h-1, a, c-1) \oplus M(V_R, h, j, c, b) : i < h \leq j, a < c \leq b\}$.
- Pour un noeud interne linéaire négatif V de T, ayant pour fils V_L et V_R, $M(V, i, j, a, b)$ est un motif de longueur maximale dans l'ensemble $\{M(V_L, i, j, a, b), M(V_R, i, j, a, b)\}$ $\cup \{M(V_L, i, h-1, c, b) \ominus M(V_R, h, j, a, c-1) : i < h \leq j, a < c \leq b\}$.

Jusqu'ici, il ne diffère en rien de l'algorithme qui utilise des arbres de séparation. Le cas nouveau est le suivant :

- Pour un noeud interne premier V de T qui est étiqueté par une permutation simple σ de taille d, et qui a pour fils $V_1, \ldots, V_d$, $M(V, i, j, a, b)$ est un motif de longueur maximale dans l'ensemble

$$\Big\{\sigma\big(\big(M(V_k, h_{k-1}, h_k - 1, c_{\sigma_k - 1}, c_{\sigma_k} - 1)\big)_{1 \leq k \leq d}\big) : \\ i = h_0 \leq h_1 \leq \ldots \leq h_d = j + 1, a = c_0 \leq c_1 \leq \ldots \leq c_d = b + 1\Big\}$$

L'idée derrière cette formule est simplement de couper l'intervalle d'indices entre i et j en d intervalles pouvant être vides $I_1, \ldots, I_d$ de gauche à droite, de même de couper l'intervalle des valeurs entre a et b en d intervalles de valeurs $A_1, \ldots, A_d$ (rangés par ordre croissant), et d'associer l'intervalle d'indices I_k à l'intervalle de valeurs A_{σ_k}. Ensuite, il suffit de σ-concaténer les plus grands motifs communs aux V_k et à π' utilisant l'intervalle d'indices I_k et l'intervalle de valeurs A_{σ_k} dans π'. Avec ces notations, on a $I_k = \{h_{k-1}, \ldots, h_k - 1\}$ et $A_k = \{c_{k-1}, \ldots, c_k - 1\}$.

Une fois les tableaux $M(V, \_, \_, \_, \_)$ remplis, il suffit de retourner le contenu de la case $M(R, 1, n', 1, n')$, R étant la racine de T : c'est un plus grand motif commun à π et π'. La preuve de ce résultat est vraiment similaire à la preuve de correction de l'algorithme de la section 4.

La grosse différence avec l'algorithme de la section 4 réside dans l'analyse de la complexité de cet algorithme. Les tableaux occupent le même espace, mais le coût du remplissage d'une case à partir des tableaux précédents est bien supérieur si on considère un noeud premier d'arité d. En effet, l'ensemble dans lequel on cherche un motif de longueur maximale dans ce cas contient non plus $\mathcal{O}(n'^2)$ mais $\mathcal{O}(n'^{2d-2})$ éléments !

Si l'on ne connaît pas de borne *a priori* sur l'arité d'un noeud premier, on peut seulement dire que $d \leq n$, et cette borne est optimale, car atteinte dans le cas où π est elle-même une permutation simple. Dans ce cas, l'algorithme que j'ai décrit a une complexité totale en $\mathcal{O}(nn'^{2n+2})$, et n'est pas polynomial. En revanche, si on s'intéresse exclusivement à des permutations π telles que l'arité des noeuds premiers dans leur arbre de décomposition est bornée par une constante d, alors l'algorithme a une complexité $\mathcal{O}(nn'^{2d+2})$, et donc est polynomial.

On a donc le résultat suivant :

Théorème 5.7. *Soit d un entier fixé. Considérons la classe R des permutations dont l'arbre de décomposition a tous ses noeuds premiers d'arité bornée par d. Alors le problème de recherche de plus grand motif commun à une permutation de R et une permutation quelconque est dans P.*

Ce théorème permet de décrire des classes de permutations plus larges que les séparables pour lesquelles on a un algorithme polynomial de recherche de plus grand motif commun. Cependant, ces classes ont beau être plus larges que la classe des permutations séparables, elles restent infiniment petites par rapport à la classe de toutes les permutations. Cela se justifie par exemple par le fait que la classe des permutations dont les arbres de décomposition ont tous leurs noeuds premiers de degré inférieur à d est incluse dans la classe des permutations évitant l'ensemble de motifs $\{\sigma : \sigma$ est une permutation simple de degré $d+1\}$. Ainsi, par le théorème de Stanley-Wilf (théorème 2.2), il y a au plus c^n telles permutations de taille n pour une certaine contante c. La question qu'on peut alors se poser est de savoir s'il existe des classes de permutations regroupant une proportion non négligeable de toutes les permutations, et pour lesquelles il existe un algorithme polynomial de recherche de plus grand motif commun. Plus humblement, on peut se demander s'il existe des classes de permutations plus larges que celles décrites dans le théorème 5.7 (même si elles restent infiniment petites) et pour lesquelles un tel algorithme existe. Ces questions restent ouvertes.

6 Conclusion

Dans ce mémoire, j'ai présenté plusieurs aspects des études que l'on peut mener sur des objets combinatoires. En particulier, je me suis intéressée aux

permutations séparables, qui sont une classe particulière de permutations à motifs exclus. Mais les thèmes de recherche qui m'ont occupée peuvent être des fils directeurs pour l'étude d'un grand nombre d'objets combinatoires.

J'ai d'abord donné plusieurs définitions, ou caractérisations, des permutations séparables, en termes de motifs exclus, de graphes, par une construction ad hoc (l'arbre de séparation), et en termes d'arbres de décomposition. J'ai aussi démontré l'équivalence de ces définitions. Parmi les caractérisations proposées, celle faisant intervenir les arbres de décomposition est nouvelle. Comme beaucoup d'autres objets d'étude, les permutations séparables sont apparues dans divers contextes, ce qui explique ce foisonnement de définitions.

J'ai ensuite travaillé sur l'énumération des permutations séparables. Leur énumération était déjà connue, mais j'en propose une preuve nouvelle, bijective, et donc beaucoup moins calculatoire que les preuves existantes, utilisant pour la plupart des arbres de génération, technique qui ne permet pas d'échapper aux calculs.

Pour terminer, je me suis intéressée aux propriétés algorithmiques des permutations séparables, vis-à-vis des problèmes de recherche d'occurrence d'un motif dans une permutation, et de recherche d'un plus grand motif commun à deux permutations. On sait que dans le cas général, ces deux problèmes sont NP-durs. Il était connu aussi que la recherche d'une occurrence d'un motif séparable dans une permutation générale est dans P. J'ai démontré au cours de mon stage que la recherche d'un plus grand motif commun à une permutation séparable et une permutation générale est aussi dans P. En outre, ce résultat peut être étendu des permutations séparables à des classes de permutations plus larges : celles dont l'arbre de décomposition a tous ses noeuds premiers d'arité bornée par une constante d. Ce résultat ouvre des perspectives pour des travaux futurs, visant à mieux cerner les classes de permutations pour lesquelles le problème de recherche de plus grand motif commun est dans P.

Mais mes activités au cours du stage ne se sont pas limitées à l'étude des permutations séparables. J'ai travaillé parallèlement, et sous la houlette de chercheurs avisés, sur d'autres thèmes, principalement les mots bicoloriés et les cartes eulériennes et unicursales. Le travail sur les mots bicoloriés est bien engagé, ce qui me permet de le présenter dans l'annexe du présent rapport. Quant aux cartes, c'est un travail en cours, et qui est encore trop loin de l'aboutissement pour me permettre de le faire figurer ici.

A Annexe : Mots bicoloriés

Dans cette section, je décris le problème de bicoloriage sur lequel nous avons travaillé avec Dominique Rossin et Stéphane Vialette au début de mon stage, un peu en marge du sujet principal du stage. Je le fais pourtant figurer dans le rapport, car il a le mérite de s'énoncer facilement, de donner lieu à des solutions simples pour des sous-problèmes, et à des reformulations du problème général en des problèmes susceptibles d'intéresser une plus large communauté. Ceci nous a permis d'utiliser des résultats existants pour trouver une solution approchée au problème général.

A.1 Le problème de bicoloriage

Une instance du problème de bicoloriage est un mot de longueur paire muni d'une structure supplémentaire qui est décrite juste après. Le but du problème est de colorier les lettres du mot avec deux couleurs (rouge et vert, ou bien rond et carré) de sorte qu'en lisant le mot de gauche à droite, on minimise le nombre de changements de couleur. Bien sûr, le coloriage doit vérifier certaines contraintes, explicitées plus loin.

Définition A.1. *Un* mot w *est une séquence de longueur paire (notée* $2n$*) sur l'alphabet* $\{a, b\}$*, où les lettres sont regroupées deux par deux pour former* n *paires* (w_i, w_j)*,* $i < j$*, chaque lettre apparaissant dans une et une seule paire.*
Dans chacune de ces paires, le premier élément est un a *et le deuxième un* b*.*
Les paires sont représentées par des arcs reliant les deux éléments de la paire.

La partie gauche de la figure 16 présente un exemple de mot, instance du problème de bicoloriage.

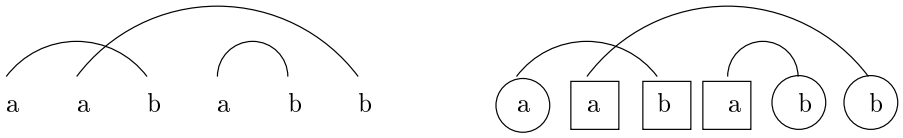


FIG. 16 – Une instance du problème de bicoloriage et un coloriage légal associé

Définition A.2. *Un* coloriage *d'un mot attribue à chaque lettre du mot une couleur : rouge (représentée par un rond) ou vert (par un carré).*
Un coloriage est dit légal *si pour chaque arc du mot, les deux extrémités de l'arc ont des couleurs différentes.*

La partie droite de la figure 16 présente un coloriage légal du mot de l'exemple précédent.

Étant donné un mot w de longueur $2n$ et un coloriage légal de ce mot, on dit qu'il y a un changement de couleur à la position i, $2 \leq i \leq 2n$, lorsque le coloriage attribue des couleurs différentes à w_{i-1} et à w_i.

On peut maintenant énoncer le problème qui nous intéresse dans toute la suite de cette section :

Problème A.1. *Problème de bicoloriage*

– INPUT : *un mot w au sens de la définition A.1.*
– OUTPUT : *un coloriage légal de w qui minimise le nombre de changements de couleur (on dira qu'un tel coloriage est* optimal*).*

On peut dès à présent distinguer des sous-problèmes du problème A.1, selon la manière dont les arcs du mot en entrée s'imbriquent entre eux.

Considérons un mot w et deux arcs (w_i, w_j) et (w_p, w_q) avec $i < p$. Il n'y a alors que trois manières possibles pour ces deux arcs de se positionner l'un par rapport à l'autre. Ces trois configurations sont représentées sur la figure 17.

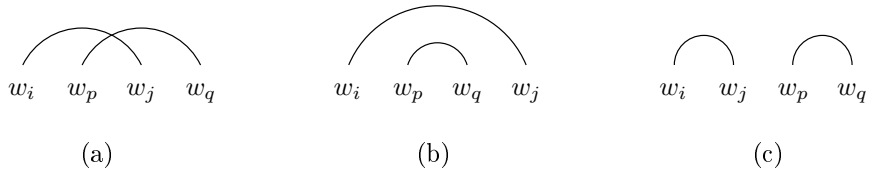


FIG. 17 – Les trois types de paire d'arcs : arcs croisés (a), arcs emboîtés (b) et arcs successifs (c)

Dans l'ensemble des mots, on peut alors distinguer trois sous-ensembles, en interdisant un des trois types d'arcs :
– les mots *croisés* : ceux dont toute paire d'arcs constitue soit des arcs successifs soit des arcs croisés,
– les mots *emboîtés* : ceux dont toute paire d'arcs constitue soit des arcs successifs soit des arcs emboîtés,
– les mots *dégénérés* : ceux dont toute paire d'arcs constitue soit des arcs emboîtés soit des arcs croisés.

Ceci donne lieu à trois sous-problèmes du problème A.1 : le problème de bicoloriage des mots croisés (resp. emboîtés, resp. dégénérés), qui est identique au problème A.1, sauf que le mot w donné en entrée est un mot croisé (resp. emboîté, resp. dégénéré).

On va voir que pour ces trois sous-problèmes, on peut calculer un coloriage minimisant le nombre de changements de couleur en temps polynomial, alors que c'est impossible dans le cas général.

A.2 Les performances de l'algorithme glouton

A.2.1 Description de l'algorithme

L'algorithme glouton prend en entrée un mot, et calcule un coloriage légal de ce mot de la manière suivante : il colorie la première lettre en rouge, puis il colorie les lettres du mot de gauche à droite, en ne changeant de couleur que lorsqu'il y est forcé. C'est-à-dire qu'il ne change de couleur que s'il rencontre la deuxième extrémité d'un arc dont la première extrémité a été coloriée avec la couleur courante. Remarquons que, par conséquent, les changements de couleur dans un coloriage calculé par l'algorithme glouton se produisent uniquement à des positions i telles que w_i est la deuxième extrémité d'un arc.

Le coloriage calculé par l'algorithme glouton est toujours légal, mais minimise-t-il le nombre de changements de couleur ? Nous allons voir que c'est le cas pour les trois problèmes de bicoloriage de mots croisés, emboîtés, ou dégénérés, mais

Algorithm 2 L'algorithme glouton de coloriage

```
INPUT : A word w of length 2n
col ← red
for j = 1 to 2n do
  if w_j is the second extremity of an arc (w_i, w_j) then
    if the color of w_i is red then
      col ← green
    else
      /*the color of w_i is green*/
      col ← red
    end if
  end if
  Color letter w_j with color col
end for
```

pas pour le problème général. L'algorithme glouton étant clairement polynomial (linéaire en la taille du mot), ceci démontrera bien que pour un mot croisé, emboîté ou dégénéré, on peut calculer un coloriage optimal en temps polynomial.

A.2.2 Le cas des mots croisés

Propriété A.3. *Pour tout mot croisé, dans le coloriage calculé par l'algorithme glouton, il y a un unique changement de couleur entre les deux extrémités d'un arc.*

Démonstration. Raisonnons par l'absurde.

Supposons qu'il existe w un mot croisé dont le coloriage par l'algorithme glouton ne vérifie pas la propriété. On dira qu'un arc de w est mauvais s'il y a strictement plus d'un changement de couleur entre les deux extrémités de cet arc.

Parmi les arcs mauvais, considérons celui dont la deuxième extrémité est la plus à gauche dans w : w_i. Supposons, sans perte de généralité, que w_i est colorié en vert par l'algorithme glouton. Il est facile de voir que w_{i-1} et w_i sont de couleur différente (sinon, en considérant le plus petit indice k tel que toutes les lettres d'indices entre k et i sont vertes, w_k serait la deuxième extrémité d'un arc, et on contredirait soit la minimalité de w_i soit on aurait une paire d'arcs emboîtés dans le mot).

On peut donc écrire $w = w^1(r^1 w_j r^2)w^2 v^1 r^3 (w_i v^2) w^3$ où w_j est la lettre appariée à w_i, donc de couleur rouge.

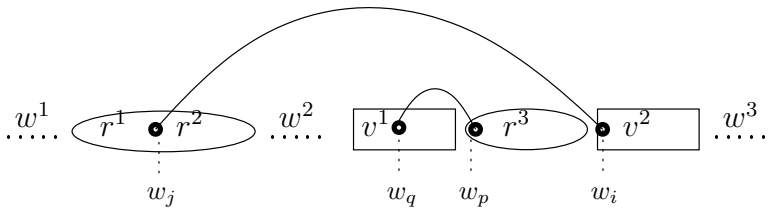


FIG. 18 – Décomposition du mot w

Dans cette décomposition, les sous-séquences $(r^1 w_j r^2)$ et r^3 (resp. v^1 et $w_i v^2$) ont toutes leurs lettres de couleur rouge (resp. verte) et sont maximales pour ce critère.

Considérons à présent la première lettre de r^3, notée w_p. Un changement de couleur se produit à cet endroit. Le coloriage étant obtenu par l'algorithme glouton, il est nécessaire que w_p soit la deuxième extrémité d'un arc. Comme w_p est à gauche de w_i, l'arc en question n'est pas mauvais. Il y a donc exactement un changement de couleur entre les deux extrémités de cet arc. La première extrémité w_q de l'arc se trouve donc dans v^1.

On observe alors que les arcs (w_q, w_p) et (w_j, w_i) sont emboîtés, d'où la contradiction qui achève la preuve. □

Conséquence A.4. *L'algorithme glouton calcule un coloriage optimal pour les mots croisés.*

Démonstration. Dans un coloriage légal, il y a au moins un changement de couleur entre les deux extrémités d'un arc. L'algorithme glouton sur un mot croisé fournissant un coloriage avec un unique changement de couleur entre les deux extrémités de chaque arc calcule donc un coloriage qui minimise le nombre de changements de couleur. □

A.2.3 Le cas des mots emboîtés

Remarquons d'abord qu'un mot emboîté est seulement un mot de parenthèses (ou mot de Dyck), où a correspond à '(' et b à ')', et où on ajoute un arc qui relie une parenthèse ouvrante à sa parenthèse fermante associée. Les arcs ne contiennent donc pas d'information : on peut les reconstruire à partir de la seule suite des lettres du mot.

Comme annoncé plus haut, on a la propriété suivante :

Propriété A.5. *L'algorithme glouton calcule un coloriage optimal pour les mots emboîtés.*

Démonstration. Considérons un mot emboîté w de longueur $2n$ et le coloriage $\mathcal{G}$ de w calculé par l'algorithme glouton. Considérons aussi un coloriage légal quelconque de w qui colorie w_1 en rouge (sinon, il suffit d'inverser toutes les couleurs), noté $\mathcal{C}$. On va montrer que le nombre de changements de couleur dans $\mathcal{C}$ est supérieur ou égal au nombre de changements de couleur dans $\mathcal{G}$.

Pour cela, on décrit une procédure qui transforme le coloriage $\mathcal{C}$ en un autre coloriage légal en faisant décroître (au sens large) le nombre de changements de couleur. En répétant cette procédure, on finit par atteindre le coloriage $\mathcal{G}$ (en au plus n applications de la procédure), démontrant ainsi que $\mathcal{C}$ a un nombre de changements de couleur supérieur ou égal à celui de $\mathcal{G}$.

La procédure est la suivante : si $\mathcal{C}$ est différent de $\mathcal{G}$, alors on considère la première lettre sur laquelle ils diffèrent, disons w_i.

Comme les deux coloriages sont légaux, si $\mathcal{C}$ et $\mathcal{G}$ diffèrent sur un b, ils diffèrent aussi sur le a associé, qui apparaît donc avant le b dans w. Ainsi, i étant minimal, $w_i = a$. Considérons le b associé à w_i, disons w_j.

Comme $\mathcal{G}(w_1) = \mathcal{C}(w_1) =$ rouge, $i \neq 1$. On peut donc parler de w_{i-1}. Remarquons que dans $\mathcal{C}$, étant donné que w_i est la première lettre sur laquelle

$\mathcal{C}$ et $\mathcal{G}$ diffèrent, $\mathcal{C}(w_{i-1}) = \mathcal{G}(w_{i-1})$ et $\mathcal{C}(w_i) \neq \mathcal{G}(w_i)$. Or $\mathcal{G}$ étant le coloriage glouton, $\mathcal{G}(w_{i-1}) = \mathcal{G}(w_i)$, puisque $w_i = a$. On a donc $\mathcal{C}(w_i) \neq \mathcal{C}(w_{i-1})$.

Ces remarques faites, définissons un nouveau coloriage $\mathcal{C}'$ légal par :

- pour $k < i$, $\mathcal{C}'(w_k) = \mathcal{C}(w_k)$
- pour $i \leq k \leq j$, $\mathcal{C}'(w_k) \neq \mathcal{C}(w_k)$
- pour $k > j$, $\mathcal{C}'(w_k) = \mathcal{C}(w_k)$

c'est-à-dire qu'on inverse le coloriage à l'intérieur de la parenthèse (w_i, w_j). Comparons maintenant le nombre de changements de couleur dans les coloriages $\mathcal{C}$ et $\mathcal{C}'$. Les seules positions où une différence de *changement* de couleur peut se produire entre $\mathcal{C}$ et $\mathcal{C}'$ sont les positions i et $j+1$ (si $j \neq 2n$). Comme on l'a remarqué plus haut, il existait dans $\mathcal{C}$ un changement de couleur à la position i et il a disparu dans $\mathcal{C}'$. En contrepartie, il est possible qu'un changement de couleur apparaisse en position $j+1$ dans $\mathcal{C}'$, qui n'existait pas dans $\mathcal{C}$. Le nombre de changements de couleur peut donc décroître de 1 ou de 2 ou rester constant lorsque l'on passe de $\mathcal{C}$ à $\mathcal{C}'$.

Enfin, il est clair que, partant de $\mathcal{C}$ et en répétant cette procédure, on finit par atteindre $\mathcal{G}$ au bout de au plus n répétitions.

On conclut donc que pour tout coloriage légal $\mathcal{C}$, le nombre de changements de couleur dans $\mathcal{C}$ est supérieur ou égal à celui dans $\mathcal{G}$. Ainsi, le coloriage calculé par l'algorithme glouton minimise le nombre de changements de couleur.

□

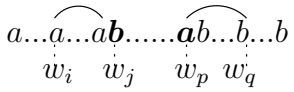
A.2.4 Le cas des mots dégénérés

Ce cas est le plus facile. En effet, le coloriage calculé par l'algorithme glouton est toujours le même sur un mot dégénéré !

Propriété A.6. *Un mot dégénéré de longueur $2n$ est toujours de la forme $a^n b^n$*

Démonstration. Raisonnons par l'absurde.

Supposons donc qu'il existe un mot dégénéré w de longueur $2n$ qui ne soit pas de la forme $a^n b^n$. Il y a donc un $\boldsymbol{b}$ et un $\boldsymbol{a}$ dans w tel que ce $\boldsymbol{b}$ apparaisse avant ce $\boldsymbol{a}$:



C'est-à-dire qu'il existe deux indices $j < p$ tels $w_j = \boldsymbol{b}$ et $w_p = \boldsymbol{a}$. Les a (resp. b) correspondant aux premières (resp. deuxièmes) extrémités des arcs, il existe des indices $i < j$ et $q > p$ tels que $w_i = a$, $w_q = b$ et (w_i, w_j) et (w_p, w_q) forment des arcs. Comme $i < j < p < q$, ces deux arcs sont successifs, ce qui contredit le fait que w est un mot dégénéré. □

Conséquence A.7. *L'algorithme glouton calcule un coloriage optimal pour les mots dégénérés.*

Démonstration. D'après la propriété précédente, il est clair que le coloriage optimal d'un mot dégénéré de longueur $2n$ est de colorier les n premières lettres en rouge et les n suivantes en vert. Et c'est ce coloriage que calcule l'algorithme glouton. □

A.2.5 Le cas général

Au vu des résultats précédents, on serait tenté de croire que l'algorithme glouton appliqué à n'importe quel mot fournit un coloriage optimal. Et pourtant ! On peut exhiber des exemples de mots de longueur $2n$ où un coloriage optimal fait un nombre constant de changements de couleur, et où l'algorithme glouton induit de l'ordre de n changements de couleur. Ceci démontre d'une part que l'algorithme glouton ne calcule pas un coloriage optimal, mais qu'il ne fournit pas non plus une approximation d'un tel coloriage : il n'existe pas de constante k telle que le nombre de changements de couleur dans le coloriage glouton soit inférieur à k fois le nombre de changements de couleur dans un coloriage optimal.

Revenons à l'exemple de la figure 16. Dans ce mot, il y a trois arcs, donc trois paires d'arcs, et elles sont chacune d'un type différent. Sur la figure 19, on compare le coloriage glouton (à gauche) et le coloriage optimal (à droite) de ce mot.

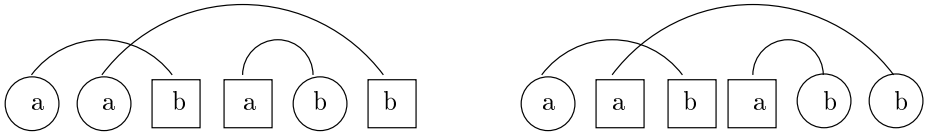


FIG. 19 – Coloriage glouton et coloriage optimal

Dans le coloriage glouton, il y a trois changements de couleur (aux positions 3, 5 et 6), alors qu'il n'y en a que deux dans le coloriage optimal (aux positions 2 et 5).

En partant de cet exemple on peut construire des mots de longueur $4n + 6$, pour tout $n \geq 1$, où il y a 2 changements de couleur dans un coloriage optimal, et $2n + 2$ dans le coloriage glouton. Pour cela, on ajoute des "paquets de lettres" comme illustré sur la figure 20 : en gras on retrouve les arcs de l'exemple initial, où sont intercalés deux blocs de n a, dont les b associés se trouvent à la fin du mot, avec en alternance un b provenant du premier bloc de a et un provenant du deuxième.

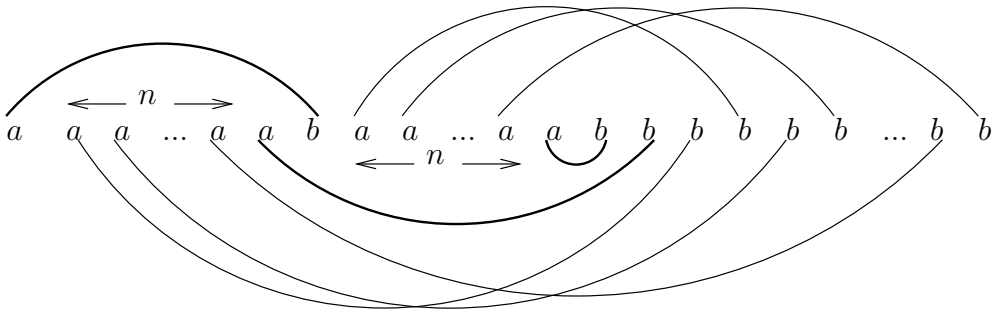


FIG. 20 – L'exemple pathologique

La figure 21 montre les deux coloriages (glouton en haut, optimal en bas)

que l'on peut obtenir sur l'exemple de la figure 20 : dans le coloriage glouton, les deux blocs de n a insérés sont coloriés par des couleurs différentes, ce qui provoque à la fin du mot une alternance de couleur sur les $2n$ dernières lettres qui sont les b correspondant à tous ces nouveaux a insérés.

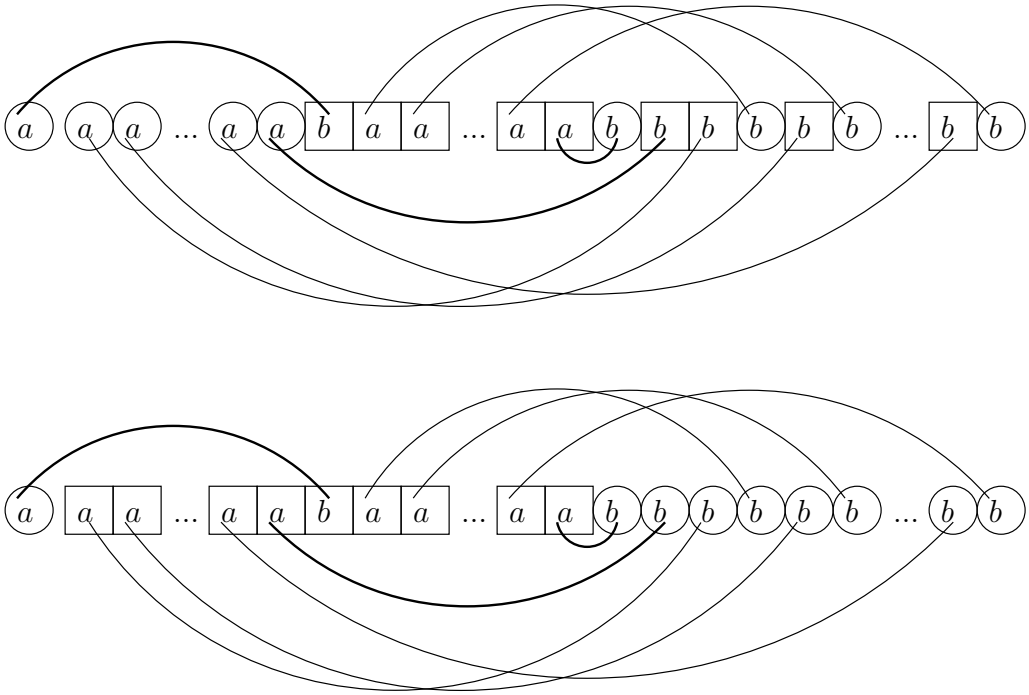


FIG. 21 – Coloriage glouton et coloriage optimal sur l'exemple pathologique

On voit donc bien sur cet exemple à quel point l'algorithme glouton est mauvais quand il s'agit de trouver un "bon" coloriage d'un mot (i.e. un coloriage dont le nombre de changements de couleur est proche de l'optimal). Il est alors naturel de se demander s'il existe d'autres algorithmes polynomiaux qui calculent des coloriages légaux dont on peut garantir qu'ils sont une bonne approximation de l'optimal. C'est à ces problèmes d'approximation qu'on s'intéresse dans la sous-section suivante.

A.3 Complexité du problème de bicoloriage dans le cas général, et approximation d'une solution

A.3.1 Résultats de complexité

Le problème de bicoloriage général décrit au début est en fait un sous-problème du *paint shop problem* [8, 17]. Pour passer du *paint shop problem* au problème de bicoloriage, on restreint les instances autorisées en entrée du problème. Th. Epping, W. Hochstättler et P. Oertel ont démontré [17] que le *paint shop problem* est $\mathcal{NP}$-complet. Paul Bonsma [8] a montré par la suite que le problème restreint de bicoloriage est aussi $\mathcal{NP}$-complet, et qu'il est en

outre $\mathcal{APX}$-dur. La définition suivante rappelle ce qu'est un algorithme approché pour un problème de minimisation tel que le problème de bicoloriage, et définit la classe des problèmes $\mathcal{APX}$ et la notion de problème $\mathcal{APX}$-dur.

Définition A.8. *Un algorithme (pour un problème de minimisation) est* ρ-approché *si, pour toute instance du problème dont la valeur de la solution optimale est OPT, il calcule une solution dont la valeur est au plus $\rho \times OPT$.*

Un schéma d'approximation *($\mathcal{PTAS}$) pour un problème de minimisation fournit un algorithme $(1+\epsilon)$-approché et en temps polynomial pour tout $\epsilon > 0$.*

La classe *$\mathcal{APX}$ regroupe tous les problèmes de minimisation pour lesquels il existe $\rho > 1$ et un algorithme ρ-approché et en temps polynomial pour ce problème.*

*Un problème est dit $\mathcal{APX}$-*dur *lorsque l'existence d'un schéma $\mathcal{PTAS}$ pour ce problème implique que pour tout problème de la classe $\mathcal{APX}$, un $\mathcal{PTAS}$ existe.*

Le théorème suivant [4] permet de se rendre compte à quel point il est difficile de trouver une solution approchée à un problème $\mathcal{APX}$-dur :

Théorème A.9. *Si un problème $\mathcal{APX}$-dur admet un schéma d'approximation $\mathcal{PTAS}$, alors $\mathcal{P} = \mathcal{NP}$*

Pour approcher les problèmes $\mathcal{APX}$-durs, on ne cherche donc pas de schémas $\mathcal{PTAS}$, mais plutôt (lorsqu'on y arrive) des algorithmes polynomiaux ρ-approchés, pour une constante ρ déterminée.

Pour le problème de bicoloriage, qui est $\mathcal{APX}$-dur, on ne connaît pas d'algorithme polynomial ρ-approché pour aucune constante ρ. Dans ce qui suit, on propose une méthode qui aboutit à un algorithme $\mathcal{O}(\log n)$-approché, où n est la longueur du mot en entrée du problème.

A.3.2 Les problèmes *MinMulticut* et $2CNF \equiv$*deletion*

Le problème de *MinMulticut* est une extension naturelle du problème *MinCut* de recherche d'un coupure minimale dans un graphe, où l'on n'a plus *un* couple de sommets source-destination, mais un *ensemble* de tels couples.

Problème A.2. *Problème de* MinMulticut

- INPUT :
  - *un graphe G non-orienté où chaque arête possède une capacité*
  - *un ensemble de k couples de sommets (s_i, t_i), $1 \leq i \leq k$ de G, dits couples source-destination*
- OUTPUT : *une coupure minimale, i.e. un ensemble d'arêtes dont la somme des capacités est minimale et telles que lorsqu'on les retire du graphe, on déconnecte s et t pour chaque couple source-destination (s,t)*

Dans le cas du problème *MinCut*, il est bien connu que la capacité de la coupure minimale est égale au flot maximal qui peut passer de la source s à la destination t. Ce résultat n'est plus vrai dès que l'on autorise $k \geq 2$ couples source-destination. En revanche, on a les inégalités suivantes [21] :

$$\frac{\text{multicoupure minimale}}{\mathcal{O}(\log k)} \leq \text{multiflot maximal} \leq \text{multicoupure minimale}$$

Dans [21], N. Garg, V. Vazirani et M. Yannakakis prouvent ces inégalités et donnent un algorithme polynomial de calcul d'une multicoupure dont la capacité est au plus $\mathcal{O}(\log k)$ fois celle d'une multicoupure minimale. Cet algorithme utilise des techniques de programmation linéaire : le calcul d'une multicoupure correspond à un problème de programmation linéaire en nombres entiers, et la résolution de la version relaxée fournit la $\mathcal{O}(\log k)$-approximation annoncée de la multicoupure minimale. Cet algorithme est repris par K. Munagala [29], qui représente chaque arête e du graphe comme un tuyau ayant un certain diamètre (sa capacité c_e), et une certaine longueur x_e, les x_e correspondant aux variables du programme linéaire. Dans le cas du programme linéaire en nombres entiers, les x_e valent 1 pour les arêtes qui appartiennent à la multicoupure minimale, et 0 sinon. La fonction objectif à minimiser est $\sum_e c_e x_e$, dont l'interprétation physique est le volume total de la "tuyauterie" que représente le graphe.

Une application de la résolution approchée du problème A.2 est développée dans [21] : il s'agit de fournir une solution approchée à un autre problème, celui de $2CNF \equiv$deletion.

Problème A.3. *Problème de* $2CNF \equiv$deletion
- INPUT : *un ensemble $\mathcal{E}$ de clauses de la forme $l_1 \equiv l_2$ où l_1 et l_2 sont des littéraux, i.e. des variables ou des variables niées*
- OUTPUT : *un sous-ensemble $\mathcal{E}'$ de $\mathcal{E}$ de cardinalité minimale tel que $\mathcal{E} \setminus \mathcal{E}'$ est satisfaisable*

Une instance du problème A.3 peut être transformée en une instance du problème A.2 comme suit. Notons $\{x_1, \ldots, x_n\}$ l'ensemble des variables apparaissant dans les clauses de $\mathcal{E}$. On construit un graphe dont l'ensemble des sommets est $V = \{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\}$, et dont l'ensemble des arêtes est $E = \{(l_1, l_2) : l_1 \equiv l_2 \in \mathcal{E}\} \cup \{(\neg l_1, \neg l_2) : l_1 \equiv l_2 \in \mathcal{E}\}$, avec la convention que $\neg\neg x = x$ pour toute variable x. On donne une capacité 1 à toutes les arêtes et on fixe n couples source-destination dans ce graphe : $(x_i, \neg x_i)$, $1 \leq i \leq n$. Le poids (i.e. le nombre d'arêtes dans le cas particulier de capacités unitaires) d'une coupure minimale sur ce graphe est compris entre E' et $2E'$, où E' désigne la cardinalité d'un ensemble $\mathcal{E}'$ de cardinalité minimale solution du problème A.3 sur l'instance donnée. Le facteur 2 provient du fait que chaque clause de $\mathcal{E}$ donne lieu à 2 arêtes dans le graphe.

Une conséquence directe du résultat d'approximation du problème A.2 est qu'il existe un algorithme polynomial pour calculer une solution approchée au problème A.3 de cardinalité au plus $\mathcal{O}(\log n)$ fois la cardinalité minimale, n désignant le nombre de variables.

A.3.3 Réduction du problème de bicoloriage à $2CNF \equiv$*deletion*

L'utilité d'introduire les problèmes A.2 et A.3, avec les résultats d'approximation les concernant, devrait apparaître plus clairement dans ce paragraphe. En effet, on va voir qu'il est facile de réduire le problème de bicoloriage (qui est celui qui nous intéresse au départ) au problème de $2CNF \equiv$deletion.

Considérons une instance w du problème de bicoloriage, et construisons une instance du problème de $2CNF \equiv$deletion. Pour ensemble de variables, nous choisissons $\{x_i : w_i$ est la première extrémité d'un arc de $w\}$. Remarquons qu'il

y a donc n variables, si w est de longueur $2n$. On attribue un littéral $L(i)$ à chaque lettre w_i de w :

- $L(i) = x_i$ si w_i est la première extrémité d'un arc
- $L(i) = \neg x_j$ si w_i est la deuxième extrémité de l'arc (w_j, w_i)

Les deux couleurs du coloriage correspondent aux deux valeurs de vérité $\top$ et $\bot$. Il est clair d'après le codage qu'une affectation des variables correspond à un coloriage légal de w, et réciproquement.

Définissons maintenant l'ensemble $\mathcal{E}$ des clauses par $\mathcal{E} = \{L(i) \equiv L(i+1) : 1 \leq i \leq 2n-1\}$. Étant donnée une affectation des variables, chaque clause de $\mathcal{E}$ non-satisfaite correspond à un changement de couleur dans le coloriage de w correspondant à cette affectation.

Considérons à présent un sous-ensemble $\mathcal{E}'$ de $\mathcal{E}$ tel que $\mathcal{E} \backslash \mathcal{E}'$ est satisfaisable. On construit un coloriage légal de w en mettant un changement de couleur en position i si et seulement si $L(i-1) \equiv L(i) \in \mathcal{E}'$. Réciproquement, on peut faire correspondre un ensemble $\mathcal{E}'$ à chaque coloriage légal de w. L'ensemble $\mathcal{E}'$ est de cardinalité minimale exactement lorsque le coloriage associé de w a un nombre minimal de changements de couleur.

Remarquons enfin qu'un coloriage de w est uniquement caractérisé par les positions de ses changements de couleur (modulo interversion des couleurs).

En utilisant la solution $\mathcal{O}(\log n)$-approchée que l'on peut calculer pour l'instance construite du problème A.3, on obtient un coloriage légal de w dont le nombre de changements de couleur est au plus $\mathcal{O}(\log n)$ fois l'optimal (rappelons que n est la moitié de la longueur de w) :

Propriété A.10. *On peut calculer, en temps polynomial, une solution $\mathcal{O}(\log n)$-approchée au problème de bicoloriage, si n désigne la taille du mot donné en entrée de ce problème.*

A.4 Conclusion et perspectives

Dans cette annexe, le problème étudié est le genre de problèmes que j'affectionne tout particulièrement : parce qu'il est d'un énoncé très simple, compréhensible par tous, mais qu'il permet en même temps de se poser des questions non triviales, et d'introduire des concepts importants de théorie de la complexité.

Il a été démontré récemment que le problème de bicoloriage est NP-complet, et même APX-dur. Mais aucun résultat n'existait dans la littérature concernant trois sous-problèmes naturels du problème de bicoloriage. C'est d'abord à ces sous-problèmes que nous nous sommes intéressés, en exhibant un algorithme polynomial (l'algorithme glouton) qui les résout.

Revenant ensuite au problème général, on a vu que l'algorithme glouton est loin de résoudre le problème de bicoloriage, et qu'il n'en fournit même pas une approximation. Par réductions successives, on a pu démontrer qu'il existe un algorithme polynomial $\mathcal{O}(\log n)$-approché résolvant le problème. La principale question qui reste encore ouverte est de savoir s'il existe un algorithme polynomial ρ-approché, pour une *constante* ρ, qui fournirait une solution approchée au problème de bicoloriage.

Remerciements

En tout premier lieu, je remercie chaleureusement mon encadrant au cours de ce stage, Dominique Rossin, pour sa bienveillance, sa disponibilité, et son dynamisme à toute épreuve.

Merci aussi à l'équipe Algorithmique et Combinatoire du Liafa, où j'ai trouvé un environnement de travail formidable. Au sein de cette équipe, je remercie tout particulièrement Anne Micheli et Dominique Poulalhon pour m'avoir offert de leur temps de recherche et m'avoir fait partager de leur expérience, Philippe Gambette et Fabien de Montgolfier pour m'avoir initiée à des aspects de la théorie des graphes qui ont fait avancer mon travail à pas de géants. En outre, je remercie Stéphane Vialette et Robert Cori, qui ont apporté avec eux des problèmes (et des solutions !) lors de leurs visites au Liafa.

Merci enfin à tous les participants au groupe de travail Combinatoire à l'X, pour leur accueil et pour l'ouverture que ce séminaire a apporté à ma vision très restreinte de la combinatoire. Et merci aux stagiaires et doctorants des laboratoires Liafa et PPS, et autres buveurs de thé-ou-café, qui ont créé un environnement humain agréable et convivial, où l'on n'arrive jamais à reculons le matin.

Références

[1] M.H. Albert, R. E. L. Aldred, M. D. Atkinson, H. P. van Ditmarsch, B. D. Handley, C. C. Handley, J. Opatrny, *Longest subsequences in permutations*, Australian J. Combinatorics 28 (2003), 225-238

[2] M.H. Albert, M.D. Atkinson, *Simple Permutations and Pattern restricted Permutations*, Discrete Mathematics 300 (2005) 1-15

[3] M.H. Albert, M.D. Atkinson, M. Klazar, *The enumeration of simple permutations*, Journal of Integer Sequences, Vol. 6 (2003)

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof verification and the hardness of approximation problems*, Journal of the ACM 3, 501-555 (1998)

[5] J. Bandlow, E. S. Egge, K. Killpatrick, *A Weight-Preserving Bijection Between Schröder Paths and Schröder Permutations*, Annals of Combinatorics, Volume 6, Numbers 3-4, 235 - 248 (December 2002)

[6] A. Bergeron, C. Chauve, F. de Montgolfier, M. Raffinot, *Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs*, ESA 2005, 779-790

[7] Ph. Bille, *A Survey on Tree Edit Distance and Related Problems*, Theoretical Computer Science (2005), Vol. 1-3, No. 337, 217-239

[8] P.S. Bonsma, *Complexity results for restricted instances of a paint shop problem*, Memorandum No. 1681, Department of Applied Mathematics, Faculty of EEMCS, University of Twente (July 2003)

[9] P. Bose, J. F. Buss, A. Lubiw, *Pattern matching for permutations*, Information Processing Letters 65 (1998), 277-283

[10] M. Bousquet-Mélou, *Four classes of pattern-avoiding permutations under one roof : generating trees with two labels*, Electronic Journal of Combinatorics 9 (2003)

[11] R. Brignall, *Simple Permutations*, Permutation Patterns 2006

[12] R. Brignall, S. Huczynska, V. Vatter *Decomposing simple permutations, with enumerative consequences* (June 2006)

[13] B.-M. Bui Xuan, M. Habib, C. Paul, *Revisiting T. Uno and M. Yagiura's Algorithm*, ISAAC 2005, 146-155

[14] S. Dulucq, H. Touzet, *Analysis of tree edit distance algorithms*, Combinatorial Pattern Matching, volume 2676, 83-95, Lecture Notes in Computer Science (2003)

[15] A. Ehrenfeucht, T. Harju, P. ten Pas, G. Rozenberg, *Permutations, parenthesis words, and Schröder numbers*, Discrete Mathematics 190, 259-264 (1998)

[16] S. Elizalde *Statistics on Pattern-avoiding Permutations*, Ph. D. Thesis, M.I.T. (June 2004)

[17] Th. Epping, W. Hochstättler, P. Oertel, *Complexity results on a paint shop problem*, Discrete Applied Mathematics 136 (2004) 217-226

[18] P. Erdös, G. Szekeres, *A Combinatorial Problem in Geometry*, Compositio Mathematica vol. 2 (1935), 463-470

[19] Ph. Flajolet, R. Sedgewick, *Analytic combinatorics*, livre en construction disponible à l'adresse http ://algo.inria.fr/flajolet/Publications/books.html

[20] M. R. Garey, D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, Freeman and Co, New York, 1979

[21] N. Garg, V. Vazirani, M. Yannakakis, *Approximate max-flow min-(multi)cut theorems and their applications*, 25th STOC, 698-707 (1993)

[22] J. W. Hunt and T. G. Szymanski, *A fast algorithm for computing longest common subsequences* Commun. ACM, 20(5), 350-353 (1977)

[23] L. Ibarra, *Finding pattern matchings for permutations*, Information Processing Letters 61 (1997) 293-295

[24] I. Kaplansky, *The asymptotic distribution of runs of consecutive elements*, Ann. Math. Statistics, 16 (1945), 200-203

[25] Ph. N. Klein, *Computing the edit-distance between unrooted ordered trees*, Proceedings of the 6th Annual European Symposium on Algorithms, Lecture Notes In Computer Science, Vol. 1461, 91-102 (1995)

[26] A. Marcus, G. Tardos *Excluded Permutation Matrices and the Stanley-Wilf Conjecture*, J. Combin. Th. Ser. A. 107, 153-160 (2004)

[27] A. Micheli, D. Rossin, *Edit distance between unlabeled ordered trees*, non publié (2005)

[28] F. de Montgolfier, *Décomposition modulaire des graphes : théories extensions et algorithmes*, thèse (2003)

[29] K. Munagala, *Approximation Algorithms - Tools and Techniques*, Lecture 13 : The Minimum Multicut Problem (February 22, 2002)

[30] J. W. Raymond, P. Willett, *Maximum common subgraph isomorphism algorithms for the matching of chemical structures*, Journal of Computer-Aided Molecular Design, Vol. 16, No. 7, 521-533, (juillet 2002)

[31] L. W. Shapiro, R. A. Sulanke, *Bijections for the Schröder Numbers*, Mathematics Magazine, Vol. 73, No. 5 (Dec., 2000), 369-376

[32] K. Shearer, S. Venkatesh, H. Bunke, *An efficient least common subgraph algorithm for video indexing*, Pattern Recognition 1998 (Proceedings), Vol. 2, 1241-1243

[33] N. J. A. Sloane, *The On-Line Encyclopedia of Integer Sequences*, http ://www.research.att.com/∼njas/sequences/index.html

[34] R. P. Stanley, *Increasing and Decreasing Subsequences of Permutations and Their Variants*, non publié (2005)

[35] J. M. Steele, *Variations on the monotone subsequence theme of Erdös and Szekeres*, Discrete Probability and Algorithms, D. Aldous, P. Diaconis, J. Spencer, et J. M. Steele, eds., Springer, 1995.

[36] G. Valiente, *Subtree Isomorphism and Related Problems*, Teaching at Advanced Graph Algorithms, Riga (March 28 to April 13, 2001)

[37] V. Vatter, *Enumeration schemes for restricted permutations*, Permutation patterns 2005

[38] J. West, *Generating trees and the Catalan and Schröder numbers*, Discrete Mathematics 146, 247-262 (1995)

[39] K. Zhang, J. T.-L. Wang, D. Shasha, *On the Editing Distance between Undirected Acyclic Graphs and Related Problems*, CPM 1995 : 395-407