UNIVERSITÉ LIBRE DE BRUXELLES

Faculté des Sciences
Département d'Informatique

Réarrangement de génomes, tri par inversions et distances entre permutations

Anthony LABARRE

Directeur de mémoire :
Pr. Jean-Paul Doignon

Mémoire présenté en vue de
l'obtention du grade de
licencié en Informatique

Année académique 2003-2004

Remerciements

Je voudrais remercier du fond du cœur monsieur Doignon pour les longues heures qu'il m'a consacrées, pour ses explications ainsi que pour ses relectures aussi assidues que fréquentes des différentes parties du mémoire.

Je remercie également monsieur Tannier pour avoir bien voulu me faire parvenir un extrait de sa thèse en cours de rédaction.

Enfin, j'aimerais également remercier Anne Bergeron pour ses explications.

Table des matières

1 Introduction **4**

2 Rappels **6**
2.1 Permutations 6
2.2 Notions sur les graphes 7

3 Le problème du tri par inversions **9**
3.1 Introduction 9
3.2 Complexité de MIN-SBR 11
3.2.1 Quelques notions préliminaires 11
3.2.2 Schéma de la preuve 14
3.2.3 Caractérisation des graphes de cycles 14
3.2.4 Equivalence de MAX-ECD et MAX-ACD 20
3.2.5 Complexité de MIN-SBR 25

4 La version signée du tri par inversions **28**
4.1 Permutations signées 28
4.2 Motivation 29
4.3 Quelques notions supplémentaires 30
4.4 Calcul des scores 34
4.5 Algorithmes 37
4.5.1 Recherche d'une séquence d'inversions optimale 38
4.5.2 Calcul de $d(\pi)$ 47
4.6 Applications 55

5 Etude statistique de la distance des inversions entre permutations signées **57**
5.1 Explications sur le code source 58
5.1.1 Détection des cycles 58
5.1.2 Compte du nombre de haies 60
5.1.3 Génération des permutations 60
5.2 Observations sur les permutations encadrées positives 61
5.3 Observations sur les permutations encadrées (non nécessairement positives) 71

6 Distances invariantes à droite et mesures de désordre **80**
6.1 Notations . . . 81
6.2 Définitions . . . 81
6.3 Approche groupale . . . 82
6.4 Quelques mesures de désordre . . . 84
6.4.1 Inv_2 . . . 84
6.4.2 *Max* . . . 88
6.4.3 *Exc* . . . 89
6.4.4 *Inv* . . . 90
6.4.5 *Dis* . . . 91
6.4.6 *Runs* . . . 92
6.4.7 *Osc* . . . 92
6.4.8 Remarque sur les mesures de désordre . . . 92
6.5 Adaptation de quelques distances invariantes à droite au cas signé . . . 93
6.5.1 Le groupe des isométries de l'hypercube . . . 93
6.5.2 Un autre groupe pour les permutations signées . . . 95

7 Conclusions **97**

A Code source **98**
A.1 `main.cpp` . . . 98
A.2 `permutation.h` . . . 101
A.3 `permutation.cpp` . . . 102
A.4 `pair.h` . . . 104
A.5 `pair.cpp` . . . 104
A.6 `reversal_distance.h` . . . 105
A.7 `reversal_distance.cpp` . . . 105

Chapitre 1

Introduction

Ce mémoire traite de diverses distances entre des permutations, signées ou non ; toutes ces notions seront introduites formellement par la suite, mais précisons déjà qu'une permutation signée (définition 4.1) est une permutation (définition 2.1) dans laquelle chaque élément est affecté du signe $+$ ou $-$. L'application principale considérée ici est le réarrangement de séquences, qui est un problème de bioinformatique consistant à trouver le nombre minimal d'opérations permettant de passer d'un génôme à un autre, lesdits génômes étant représentés par des permutations. Les distances ainsi évaluées entre différentes espèces peuvent être utilisées pour construire des arbres phylogéniques, représentant l'évolution entre différentes espèces à partir d'un ancêtre commun. Parmi les autres applications possibles des distances entre permutations, il y a également l'utilisation d'une certaine catégorie de distances, qui seront évoquées au chapitre 6, en statistiques et en probabilités [5], pour construire des algorithmes de tri qui tirent parti de l'ordre déjà présent dans une permutation [7], ou encore en cryptographie afin de tester si des permutations sont aléatoires [19]. Nous ne développerons pas toutes ces applications dans le mémoire - elles ne sont citées ici qu'en tant que motivations - et le lecteur intéressé est invité à consulter la bibliographie pour plus de détails.

Le travail effectué ici consiste à la fois en une synthèse d'articles et en plusieurs apports personnels. Tout d'abord, le chapitre 2 présente quelques notions de base que nous utiliserons tout au long de ce mémoire. Il s'agit de termes généraux ainsi que de quelques notions de la théorie des graphes [1]. Le chapitre 3 présente le problème dit du *tri par inversions* d'une permutation au sens classique et consiste principalement en une synthèse de [4], article dans lequel Caprara a fourni la preuve de la NP-difficulté de ce problème. Par exemple, considérons les permutations (4 1 3 2) et (2 1 4 3) ; une inversion a pour effet d'échanger les éléments sur lesquels elle agit et sera représentée par un trait sous ces éléments. Ainsi, une première inversion agissant sur (4 1 3 2) la transforme en (4 1 2 3), et une seconde inversion sur (4 1 2 3) la transforme en la seconde permutation (2 1 4 3). On dit alors que la distance des inversions entre (4 1 3 2) et (2 1 4 3) est d'au plus 2 ; elle est en fait 2 car il est impossible de transformer la première permutation en la seconde par une seule inversion. Le tri par inversions consiste à ranger les éléments d'une permutation dans l'ordre croissant (en fait, transformer la permutation donnée en la permutation identité) par le même procédé : par exemple, deux inversions suffisent à trier (4 1 3

2) (en effet ($\underline{4\ 1}\ 3\ 2$) $\rightarrow$ ($1\ \underline{4\ 3\ 2}$) $\rightarrow$ (1 2 3 4)).

Le chapitre 4, une synthèse de [2, 3, 10], présente le même problème que le chapitre 3, à ceci près qu'il est à présent envisagé dans sa version *signée* ; nous y préciserons également ce que nous entendons exactement par là, mais nous pouvons déjà faire la remarque surprenante que le problème NP-difficile du tri par inversions dans le cas non signé devient soluble en temps polynômial dans sa version signée ! Dans le cas signé, les inversions changent les signes des éléments en plus de leurs positions ; ainsi, le tri de (3 4 1 2) nécessite trois inversions : ($\underline{3\ 4}\ 1\ 2$) $\rightarrow$ ($-4\ -3\ \underline{1\ 2}$) $\rightarrow$ ($\underline{-4\ -3\ -2\ -1}$) $\rightarrow$ (1 2 3 4).

Le chapitre 5 regroupe les apports personnels que j'ai pu faire après avoir implémenté un algorithme issu de [3], permettant de calculer la distance des inversions d'une permutation : je me suis intéressé aux statistiques de la distance des inversions dans le cas signé, c'est-à-dire aux fréquences de ses valeurs en fonction du nombre d'éléments. J'ai utilisé ce code pour calculer les distances à la permutation identique de toutes les permutations signées de n éléments. Ensuite, j'ai établi des résultats de dénombrement suggérés par cette expérimentation, ce qui m'a conduit à prouver plusieurs lemmes et propositions. Enfin, le chapitre 6 consiste en une synthèse de [6, 7] et parle de distances particulières, qualifiées d'*invariantes à droite* : il existe de nombreuses autres distances, mais celles-ci, comme nous le verrons, sont particulièrement intéressantes. Leurs analogues pour le cas signé n'ont pas toujours été développés et nous présenterons une possible marche à suivre pour le faire.

Chapitre 2

Rappels

Ce chapitre présente quelques définitions générales ainsi que des notions de base de la théorie des graphes [1] qui seront utilisées tout au long de ce mémoire ; les définitions plus spécifiques, c'est-à-dire celles qui ont été introduites dans des articles, qui sont un peu plus inhabituelles ou qui ne seront utilisées qu'à un endroit bien précis du mémoire, seront introduites au fur et à mesure des besoins.

2.1 Permutations

Définition 2.1 *Une* permutation *d'un ensemble ordonné {1, 2, ..., n} de n éléments est une bijection de cet ensemble vers lui-même.*

Dans le cas où l'ensemble des n éléments est muni d'un ordre total, comme par exemple, $\{1, 2, ..., n\}$ avec l'ordre naturel, une permutation est donc un *réarrangement* des n éléments, ou encore un *arrangement* .

Les permutations seront désignées par des lettres grecques minuscules, typiquement π. Nous identifions une permutation π de $\{1, 2, ..., n\}$ avec le rangement correspondant, c'est-à-dire qu'en écrivant $\pi_i = \pi(i)$, nous identifions π au vecteur $(\pi_1\ \pi_2\ \cdots\ \pi_n)$; cette notation sera utilisée à la place de

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ \pi_1 & \pi_2 & \cdots & \pi_n \end{pmatrix}$$

La composée de deux permutations π et σ sera notée $\pi \circ \sigma$ (appliquées dans l'ordre inverse de lecture). La permutation identité $(1\ 2 \cdots\ n)$ sera notée ι. Enfin, rappelons également la décomposition en cycles bien connue, qui décompose par exemple la permutation (4 8 9 7 6 5 1 3 2 10) en les cycles (1, 4, 7), (2, 8, 3, 9) et (5, 6).

Définition 2.2 *La permutation π^{-1} est la permutation* inverse *de π, et s'obtient en échangeant positions et éléments dans π :*

$$\forall\ 1 \leq i \leq n : \quad \pi^{-1}_{\pi_i} = i$$

Cette permutation est telle que $\pi \circ \pi^{-1} = \iota$.

Définition 2.3 *Une* distance d *sur un ensemble* S *est une application* $d : S \times S \to \mathbb{R}^+ : (s,t) \mapsto r$ *vérifiant les trois axiomes suivants :*

1. $\forall\ s,t \in S : d(s,t) \geq 0$ *et* $d(s,t) = 0 \Leftrightarrow s = t$ *(caractère défini positif)*
2. $\forall\ s,t : d(s,t) = d(s,t)$ *(symétrie)*
3. $\forall\ s,t,u : d(s,u) \leq d(s,t) + d(t,u)$ *(inégalité triangulaire)*

2.2 Notions sur les graphes

Nous ne nous intéresserons ici qu'aux graphes non orientés.

Définition 2.4 *Un* graphe $G = (X,U)$ *est le couple constitué par :*

1. *un* ensemble $X = \{x_1, x_2, ..., x_n\}$;
2. *une* famille $U = \{u_1, u_2, ..., u_m\}$ *d'éléments du produit cartésien non orienté*

$$X \times X = \{\{x,y\} \mid x \in X, y \in X\}$$

Les graphes sont représentés graphiquement par des points joints par des segments ; les points sont les éléments de l'ensemble X et sont appelés *sommets*, et les segments sont les éléments de la famille U, appelés *arêtes*.

Définition 2.5 *Le nombre de sommets d'un graphe* G *est appelé l'*ordre *de* G.

Définition 2.6 *Un graphe est simple si*

1. *il ne comporte pas de boucle, c'est-à-dire d'arête dont les extrémités coïncident ;*
2. *entre deux sommets, il n'existe jamais plus d'une arête.*

Dans ce mémoire, seuls des graphes simples seront considérés.

Définition 2.7 *Deux graphes simples* $G_1 = (X_1, E_1)$ *et* $G_2 = (X_2, E_2)$ *sont* isomorphes *s'il existe une bijection de* X_1 *dans* X_2 *qui induit une bijection de* E_1 *vers* E_2.

Le fait que G_1 et G_2 sont isomorphes sera noté $G_1 \cong G_2$.

Définition 2.8 *Le* degré *d'un sommet est le nombre d'arêtes qui lui sont incidentes.*

Définition 2.9 *Deux sommets sont* adjacents *s'ils forment une arête.*

Définition 2.10 *Un graphe est* complet *si tous les couples de sommets qu'il contient sont reliés.*

Définition 2.11 *Le* complément *d'un graphe* G *est le graphe obtenu à partir du graphe complet possédant le même ensemble de sommets que* G *en retirant de ce graphe complet les arêtes de* G.

Définition 2.12 *Le* sous-graphe *de $G = (X, U)$ engendré par $A \subset X$ est le graphe dont les sommets sont les sommets de A et dont les arêtes sont les arêtes de G ayant leurs deux extrémités dans A.*

Définition 2.13 *Un graphe est* connexe *si pour toute paire de sommets distincts x, y qu'il contient, il existe un chemin entre x et y.*

Définition 2.14 *Définissons une relation d'équivalence sur les sommets, notée $x \equiv y$ qui est vraie si $x = y$ ou s'il existe un chemin entre x et y ; les classes de cette équivalence constituent une partition de l'ensemble des sommets en sous-graphes connexes de G, appelés* composantes connexes *de G.*

Chapitre 3

Le problème du tri par inversions

Ce chapitre consiste en une synthèse de [4] : nous allons y présenter le problème du tri par inversions, donner sa complexité et la prouver. Signalons que la plupart des figures de ce chapitre sont extraites de [4], et cela sera précisé quand c'est le cas.

3.1 Introduction

Définition 3.1 *Pour une permutation π de n éléments et deux indices i, j (avec $1 \leq i \leq j \leq n$), l'*intervalle *$[i \overset{\pi}{—} j]$ est le sous-vecteur*

$$(\pi_i \ \pi_{i+1} \ \cdots \ \pi_j)$$

du vecteur π.

Lorsque nous voudrons écarter une des bornes, voire les deux, les notations évidentes $]i \overset{\pi}{—} j]$, $[i \overset{\pi}{—} j[$ et $]i \overset{\pi}{—} j[$ seront utilisées. Prenons par exemple la permutation $\pi = (3\ 6\ 7\ 5\ 4\ 1\ 2)$; les intervalles $[3 \overset{\pi}{—} 5]$, $]2 \overset{\pi}{—} 7[$ et $[4 \overset{\pi}{—} 7[$ représentent respectivement les sous-vecteurs (7 5 4), (7 5 4 1) et (5 4 1).

Définition 3.2 *L'*inversion *$\rho(i, j)$ transforme toute permutation :*

$$\pi = (\pi_1 \ \cdots \ \pi_{i-1} \ \underline{\pi_i \ \pi_{i+1} \ \cdots \ \pi_{j-1} \ \pi_j} \ \pi_{j+1} \ \cdots \ \pi_n)$$

en la permutation :

$$\pi' = (\pi_1 \ \pi_2 \ \cdots \ \pi_{i-1} \ \pi_j \ \pi_{j-1} \ \pi_{j-2} \ \cdots \ \pi_{i+2} \ \pi_{i+1} \ \pi_i \ \pi_{j+1} \ \cdots \ \pi_n)$$

par une image miroir de $[i \overset{\pi}{—} j]$.

Par exemple, l'application de l'inversion $\rho(3, 7)$ à la permutation $\pi = (8\ 4\ \underline{2\ 3\ 6\ 5\ 7}\ 1\ 9)$ donne la permutation (8 4 7 5 6 3 2 1 9).

Définition 3.3 *La* distance des inversions *entre deux permutations est le nombre minimum d'inversions transformant l'une en l'autre.*

Il s'agit bien entendu de distance entre deux permutations contenant le même nombre d'éléments ; si nous choisissons $\pi_1 = (1\ 3\ 2\ 4)$ et $\pi_2 = (4\ 1\ 3\ 2)$, la distance entre ces deux permutations est de 2 : en effet, nous pouvons prendre $\rho_1(1,4)$ et $\rho_2(1,3)$, et vérifier que $(\rho_2 \circ \rho_1)\pi_2$ est bien π_1. Il s'agit bien du nombre minimum d'inversions, car aucune inversion ne permet à elle seule de transformer π_1 en π_2. Dans ce chapitre, le mot "distance" fera toujours référence à la définition 3.3. Signalons qu'il s'agit bel et bien d'une distance au sens de la définition 2.3 : en effet, il est facile de constater qu'il est toujours possible de passer d'une permutation à une autre par au moins une série d'inversions, et la distance des inversions vérifie bien les trois axiomes usuels :

1. $\forall\ \pi, \sigma : d(\pi, \sigma) \geq 0$ et $d(\pi, \sigma) = 0 \Leftrightarrow \pi = \sigma$ (trivial) ;
2. $\forall\ \pi, \sigma : d(\pi, \sigma) = d(\sigma, \pi)$; en effet, lorsque l'on dispose d'une séquence de $d(\pi, \sigma) = m$ inversions ρ_1, ..., ρ_m telles que $(\rho_m \circ \cdots \circ \rho_1)\pi = \sigma$, il suffit de les appliquer dans l'ordre inverse sur σ pour retrouver $\pi = (\rho_1 \circ \cdots \circ \rho_m)\sigma$ (cela se vérifie aisément en remarquant que $\forall\ \pi, \forall\ \rho : (\rho \circ \rho)\pi = \pi$) ;
3. $\forall\ \pi, \sigma, \tau : d(\pi, \tau) \leq d(\pi, \sigma) + d(\sigma, \tau)$; si ce n'était pas le cas, alors $d(\pi, \tau)$ ne serait pas le nombre minimal d'inversions transformant π en τ (et réciproquement), ce qui serait en contradiction avec la définition 3.3.

Le calcul de la distance entre deux permutations, tel que nous venons de l'introduire pour les permutations au sens classique, est un problème NP-difficile, et la section 3.2 se consacrera à la preuve de cette affirmation ; mais il est surprenant et intéressant de constater que la version signée de ce problème (dans laquelle les éléments permutés sont affectés de signes échangés par chaque inversion), dont nous parlerons et que nous motiverons au chapitre suivant, est polynômiale.

Le problème du tri par inversions, que nous noterons de manière plus concise MIN-SBR, est souvent reformulé comme le problème du calcul de $d(\pi)$, c'est-à-dire le nombre minimal d'inversions nécessaires pour transformer une permutation donnée π en la permutation identité ι ; c'est donc en ces termes que nous en parlerons dorénavant. Cela s'explique par l'invariance à gauche et à droite de cette distance : nous reparlerons de cette notion au chapitre 6, mais nous pouvons déjà dire qu'une distance est invariante à droite (resp. à gauche) s'il s'agit bien d'une distance et que $\forall\ \pi, \sigma, \tau : d(\sigma, \pi) = d(\sigma \circ \tau, \pi \circ \tau)$ (resp. $d(\sigma, \pi) = d(\tau \circ \sigma, \tau \circ \pi)$). Si, comme dans le cas de la distance des inversions, la distance est invariante à gauche et à droite, elle est dite *bi-invariante*. L'invariance de la distance des inversions sera montrée au chapitre 6.

Les définitions suivantes requièrent que chaque permutation π soit modifiée en l'*encadrant*, c'est-à-dire en rajoutant à ses n éléments deux nouveaux éléments $\pi_0 = 0$, $\pi_{n+1} = n+1$, sur lesquels les inversions n'agissent pas, c'est-à-dire que ces éléments resteront toujours en leurs positions respectives 0 et $n+1$ et que les indices valides pour les bornes des intervalles sur lesquels agissent les inversions resteront compris entre 1 et n.

Définition 3.4 *Un* point de rupture *d'une permutation π est un couple (π_i, π_{i+1}) tel que $|\pi_{i+1} - \pi_i| > 1$, où $0 \leq i \leq n$ (les éléments π_i et π_{i+1} sont donc voisins dans π mais pas dans ι) ; le nombre de points de rupture de π est noté $b(\pi)$. Dans le cas contraire ($|\pi_{i+1} - \pi_i| = 1$), le couple (π_i, π_{i+1}) est une* adjacence.

Définition 3.5 *Le* graphe des points de rupture *ou* graphe des cycles *associé à une permutation π de n éléments se construit de la façon suivante :*

1. *les $n+2$ éléments de la permutation encadrée π seront les sommets du graphe ;*
2. *ajouter une arête noire $\{\pi_i, \pi_{i+1}\}$ ($i = 1$, ..., n) pour chaque point de rupture (π_i, π_{i+1}) de la permutation encadrée ;*
3. *ajouter une arête grise $\{\pi_k, \pi_l\}$ pour chaque couple d'éléments (π_k, π_l) tels que $|\pi_k - \pi_l| = 1$ et $|k - l| = 1$ (donc les éléments π_k et π_l sont voisins dans ι sans l'être dans π).*

Cette structure est la raison pour laquelle l'encadrement mentionné un peu plus haut a été introduit ; de la sorte, nous obtiendrons bien des cycles dans le graphe. Pour fixer les idées, le graphe de la permutation $\pi = (0\ 8\ 7\ 1\ 4\ 5\ 2\ 6\ 3\ 9)$ est montré comme exemple à la figure 3.1.

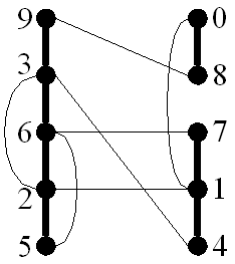


FIG. 3.1 – Graphe des cycles de $\pi = (0\ 8\ 7\ 1\ 4\ 5\ 2\ 6\ 3\ 9)$

3.2 Complexité de MIN-SBR

Nous allons à présent examiner la NP-difficulté du problème du tri par inversions, établie par Caprara [4].

3.2.1 Quelques notions préliminaires

La démonstration utilise deux problèmes auxiliaires, que nous allons introduire ici.

Définition 3.6 *Un* cycle eulérien *d'un graphe G est un cycle de G utilisant chaque arête une et une seule fois.*

Définition 3.7 *Un* graphe eulérien *est un graphe admettant un cycle eulérien.*

MAX-ECD : Ayant un graphe eulérien $G = (W, E)$, le problème appelé *MAX-ECD* (pour *MAXimum Eulerian Cycle Decomposition*) consiste à trouver une décomposition de G sous forme d'un ensemble de cycles de cardinalité maximale, donc de partitionner E en un nombre maximal de cycles.

Théorème 3.1 *[12] MAX-ECD est NP-difficile.*

Holyer établit que ce résultat est un corollaire - parmi d'autres - du résultat qu'il démontre dans son article, à savoir que le problème de partitionnement des arêtes, noté EP_n, est NP-complet pour tout $n \geq 3$; ce problème est défini de la façon suivante : étant donné un graphe $G = (V, E)$, déterminer si E peut-être partitionné en sous-ensembles E_1, E_2, ..., de telle façon que chacun des E_i génère un sous-graphe de G isomorphe (voir définition 2.7) au graphe complet K_n de n éléments.

La démonstration utilise un problème dérivé par réduction polynomiale du problème NP-complet de satisfaisabilité[1] : le problème 3-SAT[2], forcément lui aussi NP-complet. Holyer construit ensuite une réduction polynomiale du problème 3-SAT au problème EP_n, démontrant ainsi que ce dernier est NP-complet.

Définition 3.8 *Les arêtes d'un graphe* bicolore *sont partitionnées en deux ensembles disjoints, chacun caractérisé par une couleur différente.*

Dans le cadre des définitions suivantes, nous considérons un graphe $G = (V, B \cup C)$ bicolore.

Définition 3.9 *Un* cycle alterné *de G est une séquence d'arêtes b_1, c_1, b_2, c_2, ..., b_m, c_m avec :*
- $\forall\ 1 \leq i \leq m : b_i \in B$;
- $\forall\ 1 \leq j \leq m : c_j \in C$;
- *b_i et c_j ont une extrémité commune pour $1 \leq j \leq m$ avec $i = j$, et pour $\forall\ 1 \leq j \leq m$ avec $i = j + 1$ (indices pris modulo m).*

Notons que le cycle commence par une arête d'une couleur et se termine par une arête de l'autre couleur. Un exemple d'un tel cycle, si nous considèrons le graphe de la figure 3.1, est {0, 8}, {8, 9}, {9, 3}, {3, 4}, {4, 1}, {1, 0}.

Définition 3.10 *Une* décomposition en cycles alternés *d'un graphe $G = (V, B \cup C)$ (Alternate Cycle Decomposition) est une partition de $B \cup C$ en cycles alternés.*

MAX-ACD : Pour une permutation π donnée, le problème appelé *MAX-ACD* (pour *MAXimum Alternate Cycle Decomposition*) consiste à trouver une décomposition du graphe $G(\pi)$ des points de rupture de π en un ensemble de cycles alternés de cardinalité maximale.

Soit une permutation π de n éléments, et le graphe des cycles $G(\pi) = (V, B \cup C)$ correspondant où $V = \{0, \cdots, n+1\}$ est l'ensemble des sommets du graphe, correspondant à l'ensemble des éléments de la permutation encadrée π ; le graphe $G(\pi)$ est bicolore, avec B l'ensemble des arêtes noires représentant les points de rupture de π dont le nombre $|B|$ sera noté $b(\pi)$ et C l'ensemble des arêtes grises. Nous pouvons remarquer qu'il y a trois cas possibles :

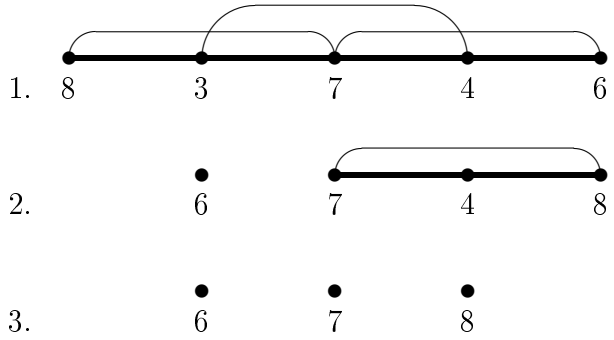
1. (π_{i-1}, π_i) et (π_i, π_{i+1}) sont des points de rupture ;

[1] Etant donné une expression booléenne sous forme normale conjonctive, donc une conjonction de disjonctions, déterminer si elle est satisfaisable.

[2] Etant donné une expression booléenne sous forme normale conjonctive et où chaque clause possède exactement trois variables, déterminer si elle est satisfaisable.

2. soit (π_{i-1}, π_i), soit (π_i, π_{i+1}) est un point de rupture ;
3. ni (π_{i-1}, π_i) ni (π_i, π_{i+1}) n'est un point de rupture.

Exemples : voici trois morceaux de graphe, illustrant respectivement chacun des cas présentés ci-dessus :



Chaque sommet i de V est donc de degré 0, 2 ou 4 (dans le premier exemple, les arêtes grises $\{2, 3\}$ et $\{4, 5\}$ n'ont pas été représentées, mais 3 et 4 sont bien de degré 4) et possède le même nombre d'arêtes incidentes grises et noires. Nous avons par conséquent $|B| = |C|$.

Une orientation[3] est parfois ajoutée aux arêtes de B : l'arête noire $\{\pi_i, \pi_{i+1}\}$ est parcourue en partant de l'élément apparaissant en premier dans π vers l'élément apparaissant ensuite.

Définition 3.11 *Un cycle alterné de $G(\pi)$ est* non orienté par rapport à π *s'il est possible de parcourir tout le cycle en suivant chaque arête noire selon son orientation, et* orienté par rapport à π *sinon.*

Le graphe de la figure 3.1 muni de l'orientation artificielle que l'on vient de définir est représenté à la figure 3.2 et comporte par exemple le cycle alterné non orienté $\{2, 6\}$, $\{6, 7\}$, $\{7, 1\}$, $\{1, 2\}$.

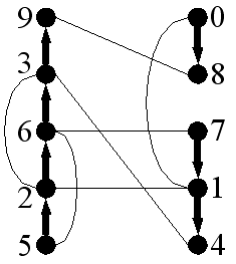


FIG. 3.2 – Graphe des cycles de $\pi = (0\ 8\ 7\ 1\ 4\ 5\ 2\ 6\ 3\ 9)$ muni de l'orientation

Nous admettons sans démonstration le résultat suivant.

Théorème 3.2 *[10] Pour une permutation π et une décomposition maximale de $G(\pi)$ en $c(\pi)$ cycles alternés, si tout cycle de la décomposition est orienté par rapport à π, alors $d(\pi) = b(\pi) - c(\pi)$.*

[3]Cette orientation étant "artificielle", dans le sens où G n'est pas un graphe orienté, je préfère conserver le terme d'*arête*, car celui d'*arc* risque d'entraîner une confusion. Pour la même raison, je conserverai le vocabulaire consacré aux graphes non orientés au lieu de sauter de l'un à l'autre continuellement.

3.2.2 Schéma de la preuve

La preuve de la NP-difficulté de la version non signée de MIN-SBR est établie en montrant une transformation polynomiale de MAX-ECD en MAX-ACD ; comme le premier problème est NP-difficile [12] (cf. Théorème 3.1), MAX-ACD l'est également. La NP-difficulté de la version non signée de MIN-SBR est ensuite établie en construisant une nouvelle transformation polynomiale, de MAX-ACD en MIN-SBR.

3.2.3 Caractérisation des graphes de cycles

Nous considérons ici une permutation $\pi = (\pi_1\ \pi_2\ \cdots\ \pi_n)$ encadrée par 0 et $n+1$, et le graphe des cycles correspondant $G(\pi)$; ayant un graphe (éventuellement bicolore) $G = (V, E)$ et un ensemble $F \subseteq \{\{i, j\} : i, j \in V, i \neq j\}$ avec éventuellement $F \not\subseteq E$, le sous-graphe de G induit par F sera noté $G\langle F\rangle$ et défini par l'ensemble de sommets $V \setminus I$ où I est l'ensemble des sommets n'étant contenus dans aucune paire de F. Si nous considérons un sous-ensemble de l'ensemble des arêtes du graphe, par exemple B ou C dans le cas d'un graphe bicolore $G = (V, B \cup C)$, les sous-graphes induits respectivement par B et C seront plutôt notés $G(B)$ et $G(C)$.

Définition 3.12 *Un graphe bicolore $G = (V, B \cup C)$ est* équilibré *si tout sommet $i \in V$ possède le même degré dans $G(B)$ et dans $G(C)$.*

Un exemple de graphe bicolore équilibré est donné à la figure 3.3.

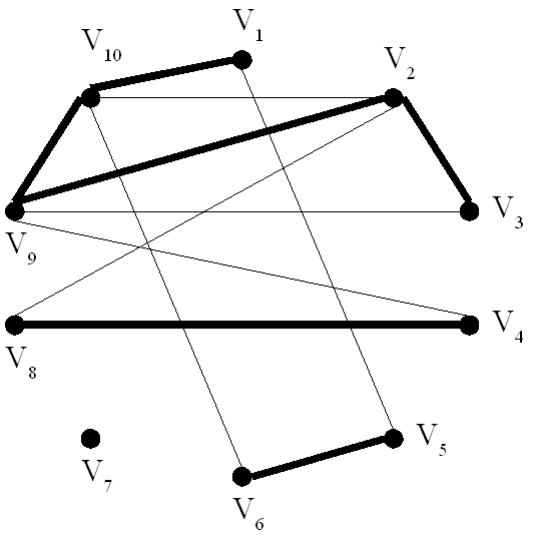


FIG. 3.3 – Un graphe bicolore équilibré

Il sera montré dans le Théorème 3.3 que le graphe des cycles d'une permutation est toujours équilibré et réciproquement que tout graphe bicolore équilibré dont les sommets sont de degré 0, 2 ou 4 est le graphe des cycles d'une certaine permutation. Rappelons que la définition 3.8 d'un graphe bicolore $G = (V, B \cup C)$ requiert $B \cap C = \emptyset$, et remarquons que la définition 3.12 implique que toute composante connexe des sous-graphes $G(B)$ et $G(C)$ est un chemin simple (c'est-à-dire un chemin ne contenant pas plusieurs fois une même arête) : par construction, les sous-graphes

$G(B)$ et $G(C)$ ne peuvent en effet pas contenir de cycles, et le degré de chaque sommet dans ces sous-graphes est 0, 1 ou 2.

Soit $G = (V, B \cup C)$ un graphe bicolore équilibré ; considérons l'ensemble U des sommets de degré 2 de G, donc l'ensemble des sommets de degré 1 dans $G(B)$ et $G(C)$. Alors $|U|$ est pair et ≥ 2.

Définition 3.13 *Une* correspondance parfaite[4] *des sommets de U est un ensemble $M \subset \{\{i,j\} : i,j \in U, i \neq j\}$ tel que tout $i \in U$ appartient à une et une seule paire de M.*

Définition 3.14 *Un* cycle hamiltonien *d'un graphe G est un cycle passant exactement une fois par chaque sommet de G.*

Définition 3.15 *Une correspondance parfaite d'un graphe bicolore $G(\pi) = (V, B \cup C)$ est* hamiltonienne *si les graphes $G\langle B \cup M\rangle$ et $G\langle C \cup M\rangle$ sont des cycles hamiltoniens.*

Illustrons les définitions précédentes par un petit exemple ; reprenons la permutation $\pi = (0\ 8\ 7\ 1\ 4\ 5\ 2\ 6\ 3\ 9)$, dont nous avions construit le graphe des cycles (figure 3.1). Une correspondance hamiltonienne de $B \cup C$ est par exemple $M = \{\{0,9\}, \{7,8\}, \{4,5\}\}$, comme cela peut être vérifié sur les sous-graphes $G\langle B \cup M\rangle$ et $G\langle C \cup M\rangle$ représentés à la figure 3.4 (les arêtes de M sont représentées en pointillés).

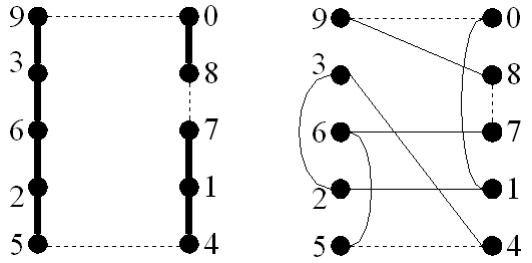


FIG. 3.4 – Respectivement, $G\langle B \cup M\rangle$ et $G\langle C \cup M\rangle$, avec $M = \{\{0,9\}, \{7,8\}, \{4,5\}\}$ et G le graphe donné en figure 3.1

Signalons qu'une illustration de la démonstration du lemme suivant suit celle-ci.

Lemme 3.1 *Soit un graphe bicolore équilibré G et U l'ensemble de ses sommets de degré 2 :*

1. *toute correspondance hamiltonienne de U définit au moins une permutation π telle que $G(\pi) \cong G$;*
2. *réciproquement, toute permutation π telle que $G(\pi) \cong G$ définit une correspondance hamiltonienne de U.*

Démonstration :

[4]Le terme utilisé par Caprara est *a perfect matching*, habituellement traduite par un *couplage parfait* en français [1] ; cependant, j'ai préféré ne pas utiliser ce terme, car un couplage parfait se base sur des arêtes du graphe, or la définition de correspondance parfaite ne requiert pas que les paires de sommets choisies soient des arêtes.

1. soit une correspondance hamiltonienne M de U ; une permutation π de $|V|$ éléments (comprenant les encadrements $\pi_0 = 0$ et $\pi_{|V|-1} = |V| - 1$) telle que $G(\pi) \cong G$ est construite comme suit :

   (a) Construisons d'abord un ensemble P d'arêtes. Si G ne possède aucun sommet de degré nul, alors $P = M$; sinon, soit i_1, ..., i_k les sommets de degré nul dans G. Choisissons arbitrairement $\{i, j\} \in M$, et prenons $P = (M \setminus \{i, j\}) \cup \{\{i, i_1\}, \{i_1, i_2\}, \ldots, \{i_{k-1}, i_k\}, \{i_k, j\}\}$. Ainsi, P est obtenu en remplaçant dans M l'arête $\{i, j\}$ par les arêtes d'un chemin de i à j parcourant tous les sommets de degré nul dans G, et $G\langle B \cup P\rangle$ ainsi que $G\langle C \cup P\rangle$ sont des cycles hamiltoniens visitant tous les sommets de G.

   (b) A présent, construisons la permutation π à partir de P. Choisissons arbitrairement un sommet $i \in U$, et parcourons le cycle hamiltonien $G\langle C \cup P\rangle$ en empruntant d'abord l'arête grise incidente à i, et numérotons les sommets ainsi parcourus de 0 (attribué à i) à $|V|-1$ suivant l'ordre dans lequel ils sont parcourus. Procédons ensuite de façon similaire avec le cycle hamiltonien $G\langle B \cup P\rangle$, en le parcourant comme précédemment à partir de i ; commençons par emprunter l'arête noire incidente à i, et numérotons les sommets de π_0 (pour i) à $\pi_{|V|-1}$ suivant l'ordre dans lequel ils sont parcourus. Il est alors facile de vérifier que $G(\pi) \cong G$, puisque nous nous sommes contenté de renuméroter les sommets de ce dernier.

2. soit une permutation π quelconque de $\{1, 2, \ldots, n\}$ telle que $G(\pi) \cong G$; prenons $P = (\{(i, i+1) : 0 \leq i \leq n\} \setminus C) \cup \{(0, n+1)\} = (\{(\pi_i, \pi_{i+1}) : 0 \leq i \leq n\} \setminus B) \cup \{(0, n+1)\}$; il est facile de vérifier que $G\langle P\rangle$ est un ensemble de chemins dont les extrémités sont les sommets de degré 2 dans G et dont les sommets intermédiaires sont ceux de degré nul dans G. De plus, l'ensemble M contenant toutes les paires $\{i, j\}$ telles qu'il y a un chemin entre i et j dans $G\langle P\rangle$ est une correspondance hamiltonienne de G. □

Illustrons le lemme 3.1 par l'exemple suivant : soit $G = (V, B \cup C)$ le graphe bicolore équilibré montré à la figure 3.3 ; $U = \{V_1, V_3, V_4, V_5, V_6, V_8\}$, et une correspondance hamiltonienne de U est $M = \{\{V_1, V_8\}, \{V_3, V_6\}, \{V_4, V_5\}\}$. Les sous-graphes correspondants $G\langle B \cup M\rangle$, $G\langle C \cup M\rangle$ sont montrés à la figure 3.5, où les arêtes de M sont dessinées en pointillés. Comme G possède un sommet de degré nul, en l'occurrence V_7, choisissons l'arête $\{V_3, V_6\}$ de M ; nous obtenons $P = \{\{V_1, V_8\}, \{V_3, V_7\}, \{V_7, V_6\}, \{V_4, V_5\}\}$. La figure 3.5 montre les sous-graphes correspondants $G\langle B \cup P\rangle$ et $G\langle C \cup P\rangle$, où les arêtes de P sont dessinées en pointillés.

Parcourons ensuite le cycle hamiltonien $G\langle C \cup P\rangle$, en prenant par exemple comme point de départ V_1, auquel nous attribuons la valeur 0, puis V_5 qui reçoit la valeur 1, et ainsi de suite ; faisons de même avec $G\langle B \cup P\rangle$, en attribuant cette fois la valeur π_0 à V_1, π_1 à V_{10} et ainsi de suite : le résultat de ces parcours est montré à la figure 3.6.

Nous pouvons alors construire le graphe $G(\pi)$ (bas de la figure 3.6), duquel nous déduisons $\pi = (0\ 7\ 3\ 8\ 4\ 5\ 6\ 1\ 2\ 9)$. Pour illustrer la deuxième partie du lemme,

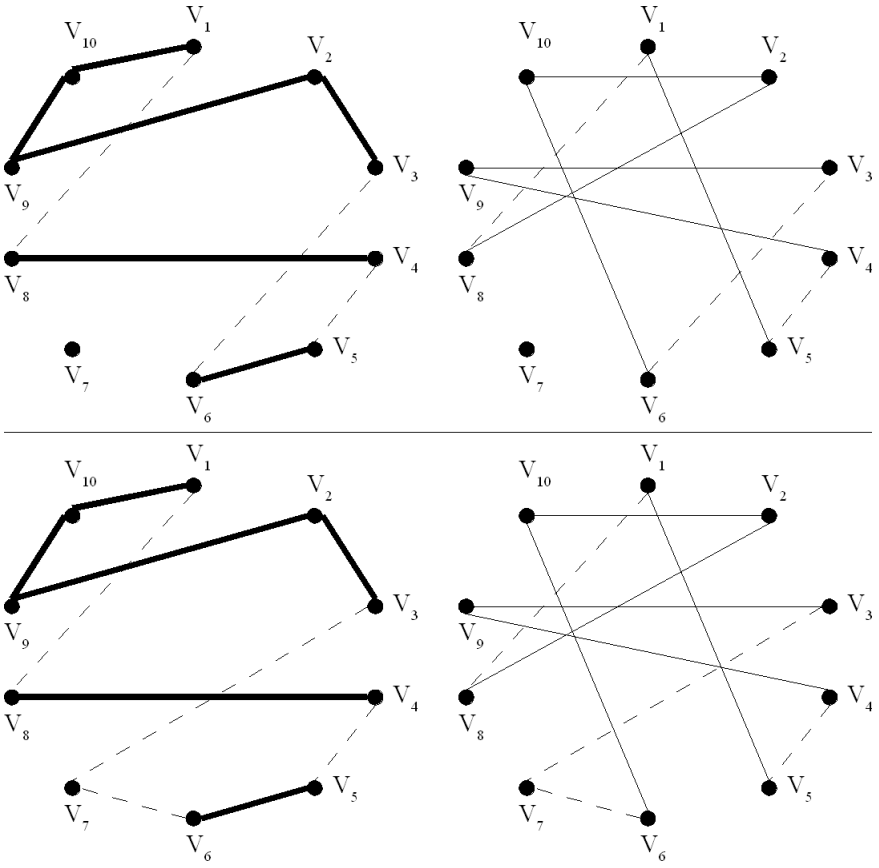


FIG. 3.5 – Au-dessus, les sous-graphes $G\langle B \cup M\rangle$ et $G\langle C \cup M\rangle$, où les arêtes de M sont en pointillés ; en-dessous, les graphes $G\langle B \cup P\rangle$ et $G\langle C \cup P\rangle$, où l'arête $\{V_3, V_6\}$ a été remplacée par $\{V_3, V_7\}$ et $\{V_7, V_6\}$ (où $G = (V, B \cup C)$ est le graphe de la figure 3.3)

nous pouvons par exemple partir de la permutation obtenue ; l'ensemble P à construire sera l'ensemble des arêtes en pointillés de la figure 3.6, et nous en déduisons une correspondance hamiltonienne M de G, qui sera l'ensemble des extrémités des chemins de P.

Définition 3.16 *Pour un graphe bicolore équilibré G, le* graphe auxiliaire $A = (U, D \cup E)$ *est construit en joignant les sommets i, j de l'ensemble U des sommets de degré 2 de G par une arête noire $d \in D$ s'il existe un chemin dans $G(B)$ de i à j, et par une arête grise $e \in E$ s'il existe un chemin dans $G(C)$ de i à j.*

Remarquons que tout sommet de A possède deux arêtes incidentes, une dans D et une dans E - mais A n'est pas nécessairement un graphe bicolore équilibré car il se peut qu'une arête de D et une arête de E possèdent les mêmes extrémités. Le graphe auxiliaire du graphe des cycles de la figure 3.1 est donné en exemple à la figure 3.7.

L'algorithme 3.1 (voir page 19) construit, sur base du graphe auxiliaire $A = (U, D \cup E)$ associé à un graphe bicolore équilibré $G = (V, B \cup C)$, une correspondance hamiltonienne des sommets de U.

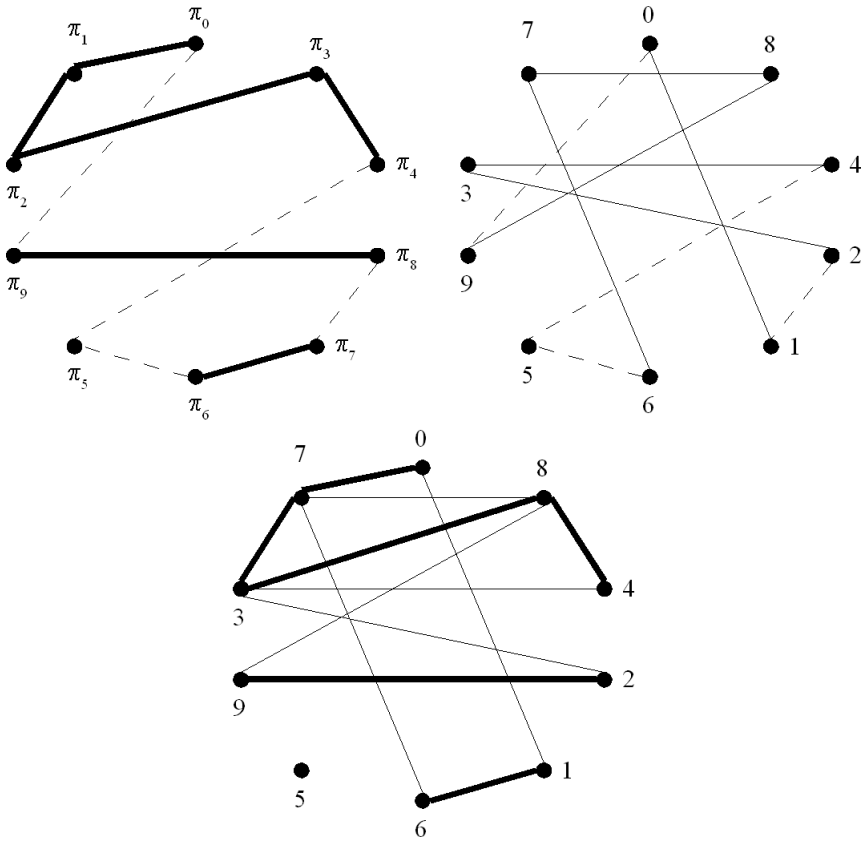


FIG. 3.6 – Les sous-graphes obtenus après les parcours respectifs de $G\langle B \cup P\rangle$ et $G\langle C \cup P\rangle$, et le graphe $G(\pi)$ duquel nous déduisons $\pi = (0\ 7\ 3\ 8\ 4\ 5\ 6\ 1\ 2\ 9)$

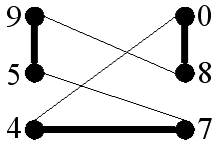


FIG. 3.7 – Graphe auxiliaire du graphe des cycles de la figure 3.1

Lemme 3.2 *L'ensemble M construit par la procédure HAMILTONIAN_MATCHING est une correspondance hamiltonienne.*

Démonstration : si $|U| > 2$, il existe toujours un couple de sommets i, j tel que $(i, j) \notin D \cup E$: il s'agit de l'ensemble des sommets de U qui sont des sommets intermédiaires dans les chemins de $G(B)$ et $G(C)$; il en découle directement que M est une correspondance parfaite des sommets de degré 2 de G telle que $M \cap (B \cup C) = \emptyset$. Pour que M soit hamiltonienne, il suffit de montrer que $G\langle B \cup M\rangle$ et $G\langle C \cup M\rangle$ ne contiennent pas de sous-cycle ; en effet, dans chaque appel récursif, il y a bijection entre les chemins maximaux de $G\langle B \cup M\rangle$ (resp. $G\langle C \cup M\rangle$) et les arêtes de D (resp. de E), donc la seule façon d'introduire un sous-cycle serait de faire correspondre deux sommets reliés par une arête de D (resp. de E), ce que nous évitons de faire. □

Algorithme 3.1 HAMILTONIAN_MATCHING(A, M)

Données: le graphe auxiliaire $A = (U, D \cup E)$ associé à un graphe bicolore équilibré $G = (V, B \cup C)$

Résultat: une correspondance hamiltonienne M des sommets de U

1: **if** $|U| = 2$ **then**
2: soit i et j les deux sommets de U ;
3: $M \leftarrow \{i, j\}$;
4: **else**
5: choisir un couple de sommets $i, j \in U$ tel que $(i, j) \notin D \cup E$;
6: soit a et b les sommets connectés respectivement à i et à j par une arête noire de D, et c et d les sommets connectés respectivement à i et à j par une arête grise de E ;
7: $U \leftarrow U \setminus \{i, j\}$;
8: $D \leftarrow (D \setminus \{\{i, a\}, \{j, b\}\}) \cup \{\{a, b\}\}$;
9: $E \leftarrow (E \setminus \{\{i, c\}, \{j, d\}\}) \cup \{\{c, d\}\}$;
10: $A \leftarrow (U, D \cup E)$;
11: HAMILTONIAN_MATCHING(A, M) ;
12: $M \leftarrow M \cup \{i, j\}$;
13: **end if**

La suppression des sommets i et j, juste avant l'appel récursif de la procédure, est illustrée à la figure 3.8 (extraite de [4]).

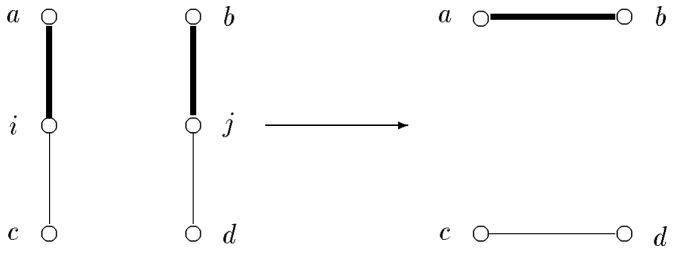


FIG. 3.8 – Suppression des sommets i et j dans la procédure HAMILTONIAN_MATCHING (source : [4]

Théorème 3.3 *Le graphe des cycles d'une permutation est bicolore équilibré. Réciproquement, tout graphe bicolore équilibré est isomorphe au graphe des cycles d'une permutation.*

Démonstration : tout graphe de cycles satisfait par définition la condition de la définition 3.12. Réciproquement, ayant un graphe bicolore équilibré G, il est possible par le lemme 3.2 de trouver une correspondance hamiltonienne de G et donc, par le lemme 3.1, une permutation π telle que $G(\pi) \cong G$. □

Définition 3.17 *Ayant un graphe bicolore $G = (V, B \cup C)$, la* subdivision *d'une arête $e = \{i, j\} \in B \cup C$ est obtenue en ajoutant deux sommets a et b à V et en*

remplaçant e par $\{i, a\}$, $\{a, b\}$ et $\{b, j\}$, où $\{i, a\}$ et $\{b, j\}$ sont de la même couleur que e et $\{a, b\}$ est de couleur différente de e.

Un exemple de cette subdivision est donné en figure 3.9, extraite de [4].

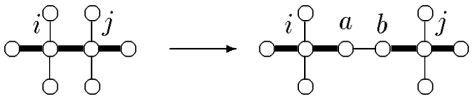


FIG. 3.9 – Subdivision de $\{i, j\}$ (source : [4])

Remarque 3.1 *Il y a bijection entre les cycles alternés (et les décompositions en cycles alternés) d'un graphe bicolore G et de n'importe quel graphe G′ obtenu à partir de G par subdivisions des arêtes.*

3.2.4 Equivalence de MAX-ECD et MAX-ACD

Caprara a montré que MAX-ECD peut être transformé en MAX-ACD et réciproquement (mais la transformation dans le sens réciproque ne nous intéresse pas).

Soit d un entier pair et $s = d/2$; le graphe bicolore $Y(d)$ est défini par :

- l'ensemble de sommets $\{b_1, ..., b_d\} \cup \{p_1, ..., p_s\} \cup \{q_1, ..., q_{s+1}\} \cup \{r_1, ..., r_s\} \cup \{t_1, ..., t_d\}$
- l'ensemble d'arêtes noires $\{\{p_i, q_i\}, \{p_{i+1}, q_{i+1}\}, i = 1, ..., s\} \cup \{\{r_i, t_{2i-1}\}, \{r_i, t_{2i}\}, i = 1, ..., s\}$
- l'ensemble d'arêtes grises $\{\{b_{2i-1}, p_i\}, \{b_{2i}, p_i\}, i = 1, ..., s\} \cup \{\{q_i, r_i\}, \{q_{i+1}, r_i\}, i = 1, ..., s\}$

Le graphe $Y(8)$, extrait de [4], est donné en exemple à la figure 3.10.

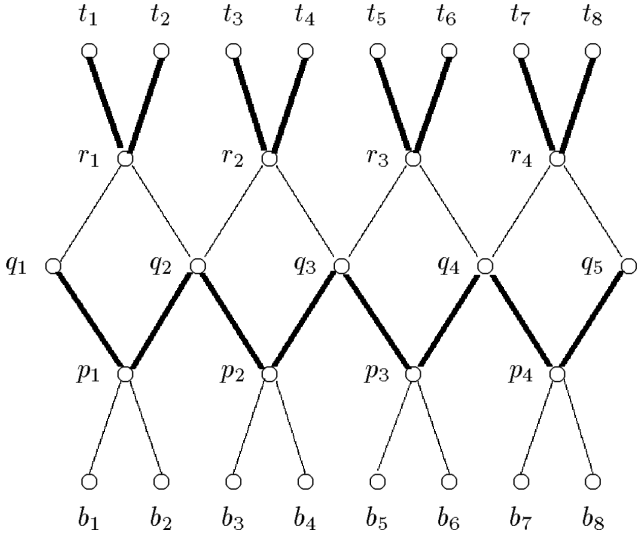


FIG. 3.10 – $Y(8)$ (source : [4])

Soit $Z(d, m)$ le graphe bicolore obtenu en fusionnant m exemplaires de $Y(d)$ (notons ces graphes $Y^1(d)$, ..., $Y^m(d)$) de la façon suivante : pour $j = 1, ..., m - 1$,

les sommets t_i de $Y^j(d)$ et les sommets b_i de $Y^{j+1}(d)$ deviennent un seul sommet dans $Z(d,m)$, qui est relié aux sommets $p_{\lceil i/2 \rceil}$ de $Y^j(d)$ et $r_{\lceil i/2 \rceil}$ de $Y^{j+1}(d)$; de plus, les sommets t_i de $Y^m(d)$ sont reliés par des arêtes grises $\{\{t_{2i-1}, t_{2i}\},\ i = 1,\ ...,\ s\}$.

Soit l_1, ..., l_d les sommets de $Z(d,m)$ correspondant aux sommets b_1, ..., b_d de $Y^1(d)$, et $b_i^j, p_i^j, q_i^j, r_i^j, t_i^j$ correspondant respectivement à b_i, p_i, q_i, r_i, t_i dans $Y^j(d)$; en particulier, pour $i = 1$, ..., m et $j = 1$, ..., $m-1$, les sommets t_i^j et b_i^{j+1} coïncident, tout comme l_1, ..., l_d et b_1^1, ..., b_d^1.

Le graphe $Z(8,2)$ est montré à la figure 3.11 (source : [4]). Les chemins alternés dans $Z(d,m)$ seront représentés par les sommets qu'ils visitent.

Définition 3.18 *Le chemin alterné* trivial *reliant les sommets* b_{2i-1}^m *et* b_{2i}^m *dans* $Z(d,m)$ *est donné par*

$$b_{2i-1}^m, p_i^m, q_i^m, r_i^m, t_{2i-1}^m, t_{2i}^m, r_i^m, q_{i+1}^m, p_i^m, b_{2i}^m$$

Pour illustrer cette définition, le chemin alterné trivial entre b_1^2 et b_2^2 est dessiné en rouge sur le graphe $Z(8,2)$, à la figure 3.11.

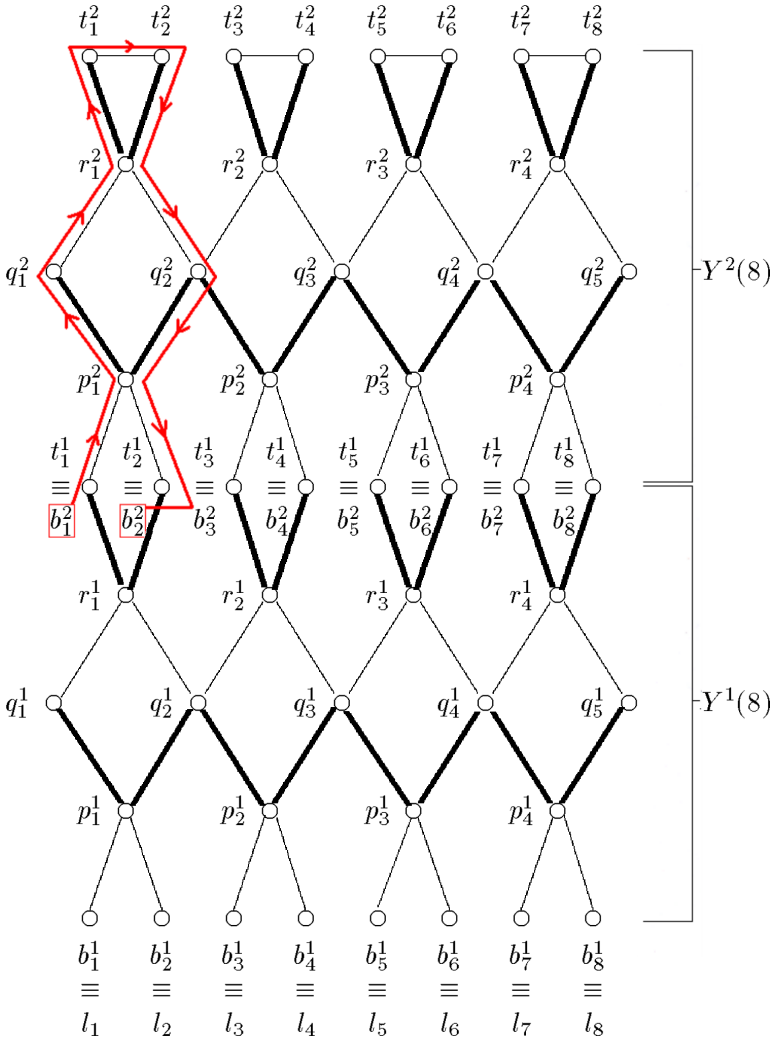


FIG. 3.11 – $Z(8,2)$ (source : [4])

Lemme 3.3 *$Z(d,m)$ ne contient aucun cycle alterné; tout sommet de $Z(d,m)$ possède le même nombre d'arêtes incidentes grises et noires, sauf l_1, ..., l_d, et tout chemin alterné reliant deux sommets de ce dernier ensemble contient exactement $8m+1$ arêtes.*

Les affirmations de ce lemme se vérifient aisément en remarquant que $Y(d)$ ne contient aucun cycle alterné, et la façon dont les m $Y^j(d)$ sont assemblés ne permet pas d'en créer dans $Z(d,m)$; de même, le nombre d'arêtes de chaque couleur incidentes à chaque sommet et le nombre d'arêtes constituant un chemin alterné découlent naturellement de la définition de $Z(d,m)$.

Lemme 3.4 *Soit τ une partition quelconque de l'ensemble de sommets $\{l_1, ..., l_d\}$ de $Z(d,m)$ en paires de sommets ; si $m \geq 1 + s(s-1)/2$, l'ensemble des arêtes de $Z(d,m)$ peut être décomposé en s chemins alternés reliant chacun un sommet d'une classe de τ à un sommet d'une autre classe.*

Démonstration : toute telle partition τ peut s'obtenir à partir de la partition $\{\{l_1,l_2\}, ..., \{l_{d-1},l_d\}\}$ en effectuant au plus $s(s-1)/2$ échanges entre des classes adjacentes ; en effet, nous pouvons obtenir la classe de π contenant l_1 en effectuant au plus $s-1$ échanges, celle contenant l_2 (si $\{l_1,l_2\}$ n'est pas déjà devenue la première classe) en effectuant au plus $s-2$ échanges, et ainsi de suite. La preuve de ce lemme s'effectue par induction sur le nombre m de graphes $Y(d)$ qui sont assemblés dans $Z(d,m)$: nous montrons que, pour $i = 1, ..., s$, il est possible de relier les sommets $l_{\tau_{2i-1}}$ et $l_{\tau_{2i}}$ par un chemin alterné reliant l_{2i-1} à l_{2i}.

1. Cas de base : si $\tau = \{\{l_1,l_2\}, ..., \{l_{d-1},l_d\}\}$, l'ensemble des arêtes de $Z(d,1)$ peut être décomposé comme requis en utilisant, pour $i = 1, ..., s$, le chemin alterné trivial reliant l_{2i-1} et l_{2i}. La figure 3.12 représente le graphe $Z(d,1)$, sur lequel nous pouvons facilement reconstituer les s chemins alternés triviaux entre l_{2i-1} et l_{2i}.
2. Induction : supposons τ obtenue à partir de τ' par un échange de deux éléments appartenant à des classes consécutives ; par hypothèse d'induction, l'ensemble des arêtes de $Z(d,j)$ peut être décomposé en s chemins alternés, le $i^{\text{ème}}$ étant de la forme $l_{\tau'_{2i-1}}$, ..., $t^j_{\tau'_{2i-1}}$, $t^j_{\tau'_{2i}}$, ..., $l_{\tau'_{2i}}$. Alors, l'ensemble des arêtes de $Z(d,j+1)$ peut être décomposé en s chemins alternés, le $i^{\text{ème}}$ étant de la forme $l_{\tau_{2i-1}}$, ..., $t^{j+1}_{\tau_{2i-1}}$, $t^{j+1}_{\tau_{2i}}$, ..., $l_{\tau_{2i}}$. Ces chemins alternés sont obtenus à partir de ceux de $Z(d,j)$ correspondant à τ' en remplaçant leurs arêtes grises $\{t^j_{2i-1}, t^j_{2i}\}$ ($i = 1$, ..., s) par des chemins alternés de $Z(j+1)$ utilisant les arêtes de $Y^{j+1}(d)$ et les arêtes $\{t^j_{2i-1}, t^j_{2i}\}$ ($i = 1$, ..., s).

   Cette transformation peut s'observer aisément sur l'exemple suivant : considérons le graphe de la figure 3.12, en prenant $d = 8$ et $j = 1$; le graphe obtenu à l'étape $j+1$ est alors celui montré en figure 3.11, dans lequel les arêtes grises $\{t^j_{2i-1}, t^j_{2i}\}$ ont été remplacées par les chemins alternés (triviaux) entre t^j_{2i-1} et t^j_{2i}).

   Il y a deux cas possibles :

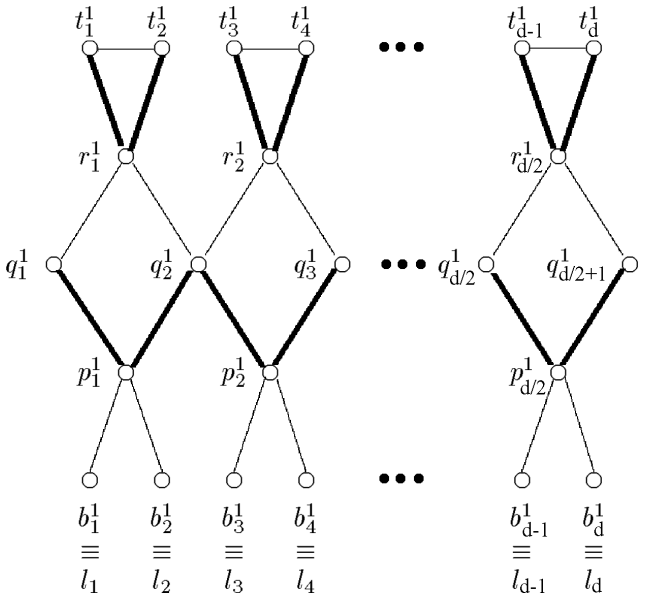


FIG. 3.12 – $Z(d,1)$

(a) les paires $\{l_{\tau'_{2h-1}}, l_{\tau'_{2h}}\}$ et $\{l_{\tau'_{2h+1}}, l_{\tau'_{2h+2}}\}$ sont remplacées dans τ par les paires $\{l_{\tau'_{2h-1}}, l_{\tau'_{2h+1}}\}$ et $\{l_{\tau'_{2h}}, l_{\tau'_{2h+2}}\}$; dans ce cas, les chemins alternés correspondant à τ sont obtenus à partir de ceux de τ' en remplaçant chaque arête $\{t^j_{2i-1}, t^j_{2i}\}$ dans $Z(d,j)$ ($i = 1, ...,h-1$ et $i = h+2, ...,s$) par le chemin alterné trivial reliant les sommets b^{j+1}_{2i-1} et b^{j+1}_{2i} dans $Z(d,j+1)$ et en remplaçant les arêtes $\{t^j_{2h-1}, t^j_{2h}\}$ et $\{t^j_{2h+1}, t^j_{2h+2}\}$ dans $Z(d,m)$ par les chemins alternés

$$b^{j+1}_{2h-1}, p^{j+1}_h, q^{j+1}_h, r^{j+1}_h, t^{j+1}_{2h-1}, t^{j+1}_{2h}, r^{j+1}_h, q^{j+1}_{h+1}, p^{j+1}_{h+1}, b^{j+1}_{2h+1}$$

et

$$b^{j+1}_{2h}, p^{j+1}_h, q^{j+1}_{h+1}, r^{j+1}_{h+1}, t^{j+1}_{2h+1}, t^{j+1}_{2h+2}, r^{j+1}_{h+1}, q^{j+1}_{h+2}, p^{j+1}_{h+1}, b^{j+1}_{2h+2}$$

L'effet de cette transformation est montré à la figure 3.13, où les arêtes en pointillés $\{t^j_{2h-1}, t^j_{2h}\}$ et $\{t^j_{2h+1}, t^j_{2h+2}\}$ sont remplacées respectivement par les chemins représentés par des disques et des rectangles.

(b) les paires $\{l_{\tau'_{2h-1}}, l_{\tau'_{2h}}\}$ et $\{l_{\tau'_{2h+1}}, l_{\tau'_{2h+2}}\}$ sont remplacées dans τ par les paires $\{l_{\tau'_{2h-1}}, l_{\tau'_{2h+2}}\}$ et $\{l_{\tau'_{2h}}, l_{\tau'_{2h+2}}\}$; la construction est alors analogue au cas précédent.

Enfin, la preuve s'achève en remarquant que si l'ensemble des arêtes de $Z(d,j)$ peut être décomposé comme requis, alors c'est également le cas pour $Z(d,j+1)$; il suffit en effet de remplacer les arêtes $\{t^j_{2i-1}, t^j_{2i}\}$ (i=1, ..., s) dans $Z(d,j)$ par les chemins alternés triviaux reliant b^{j+1}_{2i-1} et b^{j+1}_{2i} dans $Z(d,j+1)$. □

Etant donné un entier pair d, notons $m(d)$ l'entier égal à $1 + s(s-1)/2$; étant donné un graphe eulérien $H = (W,E)$, notons G_H le graphe bicolore équilibré obtenu à partir de H en remplaçant chaque sommet $i \in W$ de degré d_i par le graphe

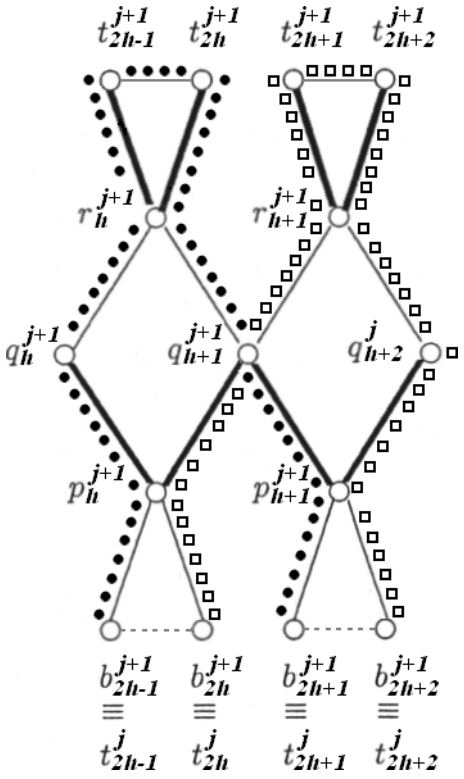


FIG. 3.13 – Premier cas du lemme 3.4

$Z(i) = Z(d_i, m(d_i))$ et puis chaque arête de E joignant deux sommets i et j de W par une arête noire joignant cette fois deux sommets de degré 1 de $Z(i)$ et $Z(j)$. Ainsi, tous les sommets de degré 1 de $Z(i)$, $(i \in W)$, possèderont exactement une arête noire incidente.

Remarquons qu'un graphe de la forme de $Y(d)$ possède exactement $\frac{7d}{2} + 1$ sommets, donc le graphe $Z(d, m)$ possède $m * (|Y(d)| - d)$ sommets (la soustraction de d provient de la coïncidence des sommets t_i^j et b_i^{j+1}), ce qui implique que l'ordre de $Z(i)$ est proportionnel à d_i^3. Une telle transformation est illustrée en figure 3.14 (source : [4]).

Théorème 3.4 *Ayant un graphe eulérien $H = (W, E)$, le problème MAX-ECD sur H peut être résolu en résolvant le problème MAX-ACD sur le graphe bicolore équilibré G_H. Ce dernier est d'ordre linéaire en celui de H et en d_H^3, où d_H est le degré maximal d'un sommet de H.*

Démonstration : la transformation de MAX-ECD en MAX-ACD, illustrée en figure 3.14, se fait de la façon décrite ci-dessus (construction de G_H). Il est alors clair (lemme 3.4) qu'étant donné une décomposition en cycles de H en p cycles, nous pouvons trouver une décomposition en cycles alternés de G_H en p cycles, et vice versa ; clairement, G_H est bicolore et équilibré. □

Les théorèmes 3.1, 3.3 et 3.4 permettent de déduire le corollaire suivant.

Corollaire 3.1 *MAX-ACD est NP-difficile.*

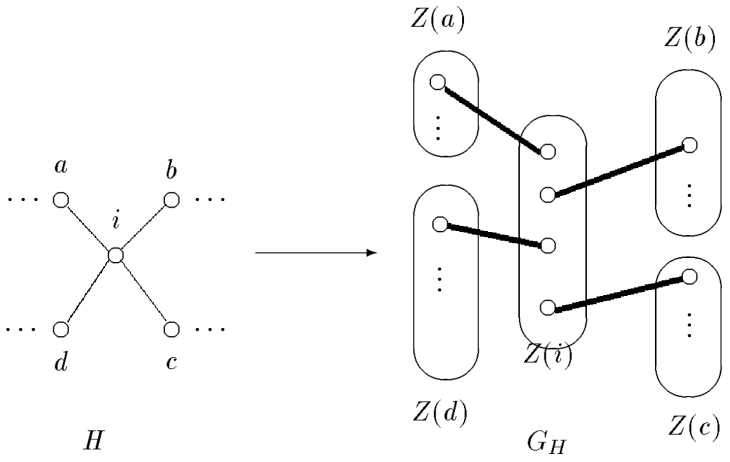


FIG. 3.14 – Transformation de MAX-ECD en MAX-ACD ; les cercles dans chacun des graphes Z représentent les sommets de degré 1 associés (source : [4])

3.2.5 Complexité de MIN-SBR

Cette section présente une transformation de MAX-ACD en MIN-SBR, montrant ainsi la NP-difficulté de ce dernier problème.

Le Théorème 3.2 peut être interprété de la façon suivante : ayant un graphe des cycles $G(\pi)$ d'une permutation π, il serait possible de calculer $c(\pi)$ en résolvant MIN-SBR sur π s'il existait une décomposition optimale en cycles alternés de $G(\pi)$ constituée uniquement de cycles orientés (par rapport à π). Malheureusement, ceci n'est pas toujours le cas ; cette difficulté va être surmontée comme suit.

Soit le graphe des cycles $G(\pi) = (V, B \cup C)$ associé à une permutation π et M la correspondance hamiltonienne de $G(\pi)$ déterminée par π ; construisons le graphe des cycles G^* et sa correspondance hamiltonienne M^* en appliquant la procédure DOUBLE_SUBDIVISION (Algorithme 3.2) à $G(\pi)$ et M ; cette procédure remplace toute arête noire de G par un chemin alterné de 5 arêtes, subdivisant "deux fois" l'arête de départ. L'effet du remplacement de l'arête noire (i, j) dans les graphes $G^*(B^* \cup M^*)$ et $G^*(C^* \cup M^*)$ est illustré à la figure 3.15, extraite de [4].

Le lemme suivant découle directement de la définition de G^* et de la Remarque 3.1.

Lemme 3.5 *L'ordre de G^* est linéaire en celui de $G(\pi)$, et il y a bijection entre les cycles alternés (et les décompositions en cycles alternés) de $G(\pi)$ et de G^*.*

Lemme 3.6 *L'ensemble M^* est une correspondance hamiltonienne de G^* et tout cycle alterné de G^* est orienté par rapport à toute permutation associée à G^* et à M^*.*

Démonstration : à la fin de l'exécution de chaque boucle `for` dans la procédure DOUBLE_SUBDIVISION, le M^* mis à jour est une correspondance hamiltonienne du nouveau G^* ; la formule permettant cette mise à jour correspond à une insertion des nouveaux sommets de degré 2 k_1, k_2, k_3, k_4 dans les cycles hamiltoniens précédents - $G^*(B^* \cup M^*)$ et $G^*(C^* \cup M^*)$. Dans $G^*(B^* \cup M^*)$, les arêtes (i, j)

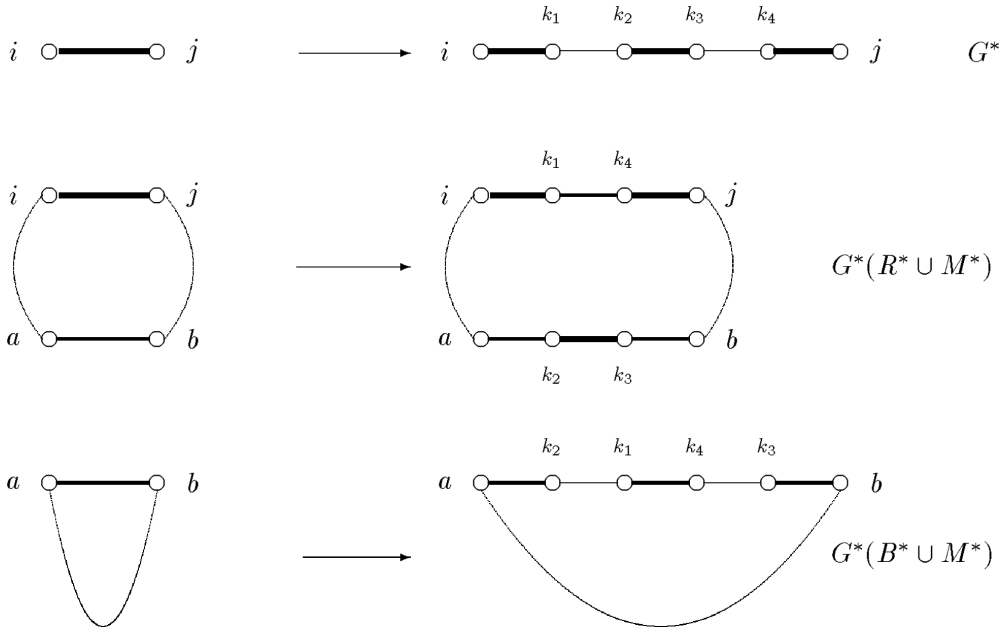


FIG. 3.15 – Remplacement de l'arête noire $\{i, j\}$ dans la procédure DOUBLE_SUBDIVISION ; les lignes pointillées sont les arêtes de M^*, et les courbes représentent des chemins.

et (a, b) sont remplacées respectivement par les chemins $\{i, k_1\}$, $\{k_1, k_4\}$, $\{k_4, j\}$ et $\{a, k_2\}$, $\{k_2, k_3\}$, $\{k_3, b\}$, alors que dans $G^*(C^* \cup M^*)$ l'arête $\{a, b\}$ est remplacée par le chemin $\{a, k_2\}$, $\{k_2, k_1\}$, $\{k_1, k_4\}$, $\{k_4, k_3\}$, $\{k_3, b\}$ (voir figure 3.15).

Toute permutation π^* associée à G^* et à M^* définit une orientation des arêtes de B^* ; en particulier, toute arête est orientée par rapport au sens dans lequel elle est traversée par le cycle hamiltonien $G^*(B^* \cup M^*)$, en commençant par le sommet correspondant à π_0^* et commençant par une arête noire.

Après le remplacement de l'arête $r = \{i, j\}$, à chaque cycle alterné de $G(\pi)$ contenant cette arête correspond un cycle alterné de G^* qui contient les arêtes $\{i, k_1\}$, $\{k_1, k_2\}$, $\{k_2, k_3\}$, $\{k_3, k_4\}$, $\{k_4, j\}$; de plus, par rapport à chaque π^* associée à G^* et M^* :

- soit l'arête $\{i, k_1\}$ est orientée de i vers k_1, l'arête $\{k_4, j\}$ de k_4 vers j et l'arête $\{k_2, k_3\}$ de k_3 vers k_2
- soit l'arête $\{i, k_1\}$ est orientée de k_1 vers i, l'arête $\{k_4, j\}$ de j vers k_4 et l'arête $\{k_2, k_3\}$ de k_2 vers k_3

Cette propriété est maintenue tout au long de la procédure, puisque les nouveaux cycles hamiltoniens $G^*(B^* \cup M^*)$ sont obtenus en remplaçant les arêtes par des sommets dans les cycles précédents ; ainsi, à la fin de la procédure, quand toutes les arêtes noires ont été remplacées, chaque cycle alterné de G^* est orienté par rapport à toute permutation associée à G^* et M^*. □

Finalement, le lemme précédent permet de déduire le résultat qui nous intéresse :

Théorème 3.5 *MIN-SBR est NP-difficile.*

Démonstration : considérons le graphe des cycles $G(\pi)$ associé à une permu-

Algorithme 3.2 DOUBLE_SUBDIVISION(G, M, G^*, M^*)

Données: un graphe des cycles $G = (V, B \cup C)$ et une correspondance hamiltonienne M des sommets de degré 2 de G

Résultat: un graphe des cycles $G^* = (V^*, B^* \cup C^*)$ et une correspondance hamiltonienne M^* des sommets de degré 2 de G^* telle qu'il y a bijection entre les cycles alternés (et les décompositions en cycles alternés) de G et de G^*, et que tous les cycles alternés de G^* sont orientés par rapport à toute permutation π^* associée à G^* et M^*

1: $V^* \leftarrow V$;
2: $B^* \leftarrow B$;
3: $C^* \leftarrow C$;
4: $M^* \leftarrow M$;
5: **for all** $r \in B$ **do**
6: soit i et j les extrémités de r et $e = \{a, b\}$ une arête quelconque de M^*, avec a et b tels que le cycle hamiltonien $G^*(B^* \cup M^*)$ est l'union de l'arête e, d'un chemin de a à i, de l'arête r et d'un chemin de j à b, (avec éventuellement $i = a$ ou $j = b$, mais pas les deux en même temps) ;
7: $V^* \leftarrow V^* \cup \{k_1, k_2, k_3, k_4\}$;
8: $B^* \leftarrow (B^* \setminus \{i, j\}) \cup \{\{i, k_1\}, \{k_2, k_3\}, \{k_4, j\}\}$;
9: $C^* \leftarrow C^* \cup \{\{k_1, k_2\}, \{k_3, k_4\}\}$;
10: **end for**

tation π, et soit M la correspondance hamiltonienne de $G(\pi)$ correspondant à π ; construisons G^* et M^* à partir de $G(\pi)$ et M en appliquant la procédure DOUBLE_SUBDIVISION, et soit π^* une permutation associée à G^* et M^*. Par les lemmes 3.5 et 3.6 et le théorème 3.2, $d(\pi^*) = b(\pi^*) - c(\pi^*)$; $c(\pi) = c(\pi^*)$, et $b(\pi^*)$ est trivialement déterminé. Nous pouvons donc calculer la valeur optimale de MAX-ACD pour $G(\pi)$ en résolvant MIN-SBR sur π^*, d'ordre linéaire en celui de $G(\pi)$; le résultat découle ainsi du corollaire 3.1. □

Chapitre 4

La version signée du tri par inversions

Ce chapitre présente la version signée du problème dont nous avons discuté au chapitre précédent, et consiste principalement en une synthèse des articles de Bergeron [2], Bergeron, Heber et Stoye [3], Hannenhalli et Pevzner [10] et Tannier et Sagot [21].

4.1 Permutations signées

Nous considèrons à présent un ensemble de n éléments admettant chacun une de deux orientations possibles ; pour simplifier l'exposé, nous allons travailler avec l'ensemble $\{1, 2, ..., n\}$ en affectant à chacun de ses éléments le signe $+$ ou $-$. Si un élément a possède l'orientation $+$, nous utiliserons la notation abrégée a au lieu de $+a$.

Définition 4.1 *Une* permutation signée *est une permutation de {1, 2, ..., n} dans laquelle chaque élément est affecté du signe + ou −.*

Un exemple d'une telle permutation, pour $n = 5$, est $(-2\ 1\ 4\ -5\ -3)$; insistons sur le fait que parler de permutation signée ne signifie pas que chaque permutation doit posséder un certain nombre ou une certaine proportion d'éléments négatifs, mais bien que chaque élément peut ou non être négatif.

Pour alléger l'exposé, nous utiliserons dans ce chapitre et dans le suivant le terme "permutation" pour désigner une permutation signée. Les permutations non signées sont également appelées permutations "classiques", ou "au sens classique".

Les notions introduites au chapitre précédent restent toujours valables mais il va falloir en adapter quelques-unes.

Définition 4.2 *L'*inversion $\rho(i, j)$ *transforme toute permutation :*

$$\pi = (\pi_1 \ \cdots \ \underline{\pi_i \ \pi_{i+1} \ \cdots \ \pi_{j-1} \ \pi_j} \ \cdots \ \pi_n)$$

en une permutation :

$$\rho(\pi) = (\pi_1 \ \pi_2 \ \cdots \ -\pi_j \ -\pi_{j-1} \ -\pi_{j-2} \ \cdots \ -\pi_{i+2} \ -\pi_{i+1} \ -\pi_i \ \pi_{j+1} \ \cdots \ \pi_n)$$

par une image miroir de $[i \overset{\pi}{—} j]$ *et une inversion des signes des éléments de cet intervalle.*

Par exemple, l'application de l'inversion $\rho(3,7)$ à la permutation $\pi = (8\ 4\ \underline{2\ -3\ 6\ 5\ -7}\ -1\ 9)$ donne la permutation $(8\ 4\ 7\ -5\ -6\ 3\ -2\ -1\ 9)$.

De manière analogue à ce qui a été fait au chapitre précédent pour les permutations classiques, nous introduisons la distance des inversions entre permutations (signées) : il s'agit du nombre minimum d'inversions au sens de la définition 4.2 transformant une permutation en l'autre. La distance entre une permutation π et l'identité sera notée $d(\pi)$.

Le problème du tri par inversions des permutations (signées) demande de déterminer $d(\pi)$ pour π donnée et de plus de fournir une séquence de $d(\pi)$ inversions transformant π en ι.

La structure du graphe des cycles introduit en section 3.1 va être à peu près la même, mais sa construction se fait d'une manière différente.

Définition 4.3 *Pour une permutation (signée)* π *donnée, le* graphe des cycles *de* π *s'obtient comme suit :*

1. *construire la permutation* π' *non signée des nombres 0 à* $2n+1$ *en remplaçant chaque élément* π_i $(1 \leq i \leq n)$ *de* π *par la séquence* $2\pi_i - 1, 2\pi_i$ *si* π_i *est positif,* $|2\pi_i|, |2\pi_i| - 1$ *sinon ; les éléments de* π' *formeront l'ensemble des sommets du graphe ; ajouter l'encadrement par 0 et* $2n+1$ *;*
2. $\forall\ 0 \leq i \leq n$ *: ajouter les arêtes noires joignant* π'_{2i} *et* π'_{2i+1} *et les arêtes grises joignant* $2i$ *et* $2i+1$*.*

Considérons la permutation $\pi = (0\ -1\ 3\ 5\ 4\ 6\ -2\ 7)$; la permutation non signée associée est donc $\pi' = (0\ 2\ 1\ 5\ 6\ 9\ 10\ 7\ 8\ 11\ 12\ 4\ 3\ 13)$, et le graphe des cycles correspondant est montré à la figure 4.1 (source : [2]).

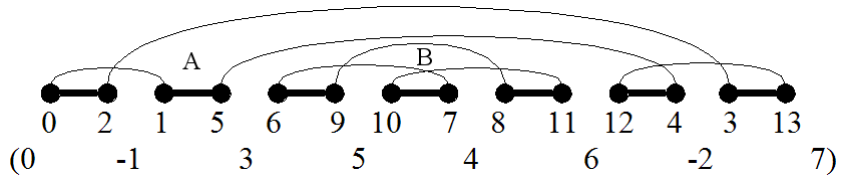


FIG. 4.1 – Graphe des cycles de $\pi = (0\ -1\ 3\ 5\ 4\ 6\ -2\ 7)$ (source : [2])

Insistons bien sur la notation π', qui désigne la permutation non signée construite à partir de π de la manière expliquée ci-dessus, car cette notation sera utilisée plusieurs fois dans la suite du chapitre.

4.2 Motivation

Les permutations sont intéressantes en bioinformatique où elles servent d'outils dans la comparaison des génômes ; ces derniers peuvent en effet être vus comme un ensemble de séquences ordonnées de gènes (en effet, chaque gène est localisé à un

endroit précis sur un chromosome), où chaque gène est donc numéroté et doté d'une orientation matérialisée dans cette modélisation par le signe + ou −.

Des espèces différentes partagent souvent le même ensemble de gènes hérités d'un ancêtre commun, mais ces gènes ont été mélangés par des mutations qui ont modifié le contenu des chromosomes, l'ensemble des gènes au sein d'un chromosome et/ou l'orientation d'un gène. Ainsi, la comparaison de deux ensembles de gènes apparaissant sur un chromosome dans deux espèces différentes peut se ramener à la comparaison de deux permutations signées et la mesure de la distance des inversions entre ces permutations fournit une bonne estimation de la divergence d'évolution entre les deux espèces. La figure 4.2 illustre l'effet d'une inversion sur un morceau de chromosome, les flèches indiquant le sens de lecture des gènes.

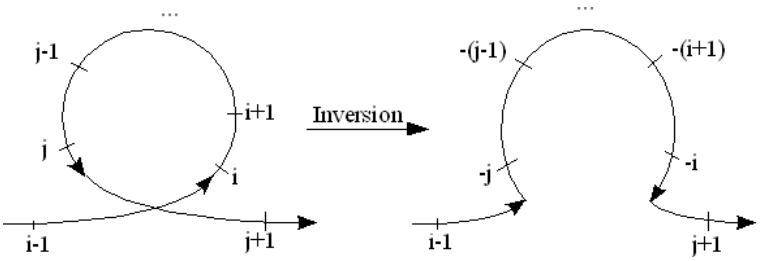


FIG. 4.2 – L'effet "physique" d'une inversion des éléments i à j d'un chromosome

4.3 Quelques notions supplémentaires

Quelques définitions nécessaires pour discuter des algorithmes décrits en section 4.5 vont à présent être introduites. Considérons une permutation signée π.

Définition 4.4 *Un couple d'éléments* (π_i, π_j) $(i < j)$ *est* orienté *si* π_i, π_j *sont de signes opposés et si* $||\pi_j|$ - $|\pi_i|| = 1$.

La permutation $\pi = (0\ -4\ -1\ 3\ 2\ 5)$ comporte quatre couples orientés : $(0, -1)$, $(-1, 2)$, $(-4, 3)$, et $(-4, 5)$.

Définition 4.5 *Ayant une permutation* π, *l'*inversion orientée *associée à un couple orienté* (π_i, π_j) *est définie par :*

$$\overline{\rho}(i,j) = \begin{cases} \rho(i, j-1) & si \quad \pi_i + \pi_j = \quad 1 \\ \rho(i+1, j) & si \quad \pi_i + \pi_j = -1 \end{cases}$$

Une propriété intéressante des inversions orientées est qu'elles rendent consécutifs les éléments des couples qui les définissent : ainsi, le résultat de l'inversion associée au couple (π_i, π_j) créera une permutation de la forme $(\cdots\ \pi_i\ \pi_j\ \cdots)$ ou $(\cdots\ -\pi_j\ -\pi_i\ \cdots)$ selon le signe de $\pi_i + \pi_j$. Par exemple, sur la permutation $\pi = (0\ -4\ -1\ 3\ 2\ 5)$, l'inversion $\overline{\rho}(-4, 3)$ donnera la permutation $(0\ -4\ -3\ 1\ 2\ 5)$, dans laquelle -4 et -3 sont à présent deux entiers consécutifs ; et l'inversion $\overline{\rho}(-4, 5)$ donnera la permutation $(0\ -1\ -2\ 3\ 4\ 5)$, où les éléments 4 et 5 apparaissent également dans le bon ordre.

Définition 4.6 *Le* score *d'une inversion orientée appliquée à une permutation π est le nombre de couples orientés dans la permutation $\rho(\pi)$ résultante.*

Si nous reprenons l'exemple précédent, les scores respectifs des inversions $\overline{\rho}(-4, 3)$ et $\overline{\rho}(-4, 5)$ sont tous deux de 2.

Hannenhalli et Pevzer [10] ont mis en évidence un paramètre "caché" qui doit être pris en compte dans le calcul de $d(\pi)$, paramètre auquel ils ont attribué le terme de *haie* en raison du fait qu'il représente un obstacle au tri de la permutation [17].

Reprenons le graphe des cycles dont nous avons parlé un peu plus tôt, pour introduire les définitions suivantes.

Définition 4.7 *Le* support *d'une arête grise du graphe des cycles d'une permutation π est l'intervalle de π' délimité par les sommets qu'elle joint, ces derniers compris.*

Définition 4.8 *Deux arêtes grises se* chevauchent *si leurs supports ont une intersection.*

Définition 4.9 *Une arête grise est* impaire *si son support contient un nombre impair d'éléments,* paire *dans le cas contraire ; une arête noire est* impaire *si ses extrémités sont de même parité,* paire *dans le cas contraire.*

Par exemple, dans le graphe de la figure 4.1, les arêtes grises $\{0, 1\}$ et $\{4, 5\}$ sont impaires et les autres arêtes grises sont paires. Les arêtes noires $\{6, 9\}$, $\{7, 10\}$ et $\{8, 11\}$ sont paires, et les autres arêtes noires sont impaires. Remarquons qu'une arête est paire si et seulement si les sommets qu'elle joint sont l'image d'un couple orienté de π.

Définition 4.10 *Une composante connexe du graphe des cycles est* impaire *si elle contient au moins une arête grise impaire,* paire *dans le cas contraire.*

Ainsi, dans le graphe de la figure 4.1, le cycle A est impair et le cycle B est pair. Nous allons maintenant introduire un nouveau graphe, construit à partir du graphe des cycles.

Définition 4.11 *Deux cycles C_1, C_2 d'un graphe des cycles $G(\pi)$ se* chevauchent *si C_1 possède une arête grise chevauchant une arête grise de C_2.*

Définition 4.12 *Soit $\mathcal{C}_\pi$ l'ensemble des cycles de $G(\pi)$; le* graphe de chevauchement des cycles *est le graphe $H_\pi = (\mathcal{C}_\pi, \mathcal{E}_\pi)$ où*

$$\mathcal{E}_\pi = \{(C_1, C_2) \mid C_1 \text{ et } C_2 \text{ se chevauchent dans } G(\pi)\}$$

Les sommets de ce graphe sont donc les cycles de $G(\pi)$, et les arêtes joignant deux sommets indiquent que les cycles correspondants se chevauchent dans $G(\pi)$.

La notion de parité attribuée aux arêtes grises et transposée aux cycles de $G(\pi)$ (définition 4.10) s'adapte naturellement au graphe H_π : une composante connexe du graphe H_π sera *impaire* si elle contient au moins un sommet impair, et *paire* dans le cas contraire.

Définition 4.13 *Pour une composante U du graphe de chevauchement des cycles, on définit :*

$$U_{min} = \min_{C \in U} \min_{\pi'_i \in C} i \text{ et } U_{max} = \max_{C \in U} \max_{\pi'_i \in C} i$$

La portée *de cette composante connexe U est* $[U_{min} \overset{\pi'}{---} U_{max}]$.

U_{min} et U_{max} sont donc les éléments de la composante U respectivement le plus à gauche et le plus à droite dans π'.

Définition 4.14 *Une composante U du graphe de chevauchement des cycles* sépare *deux composantes U' et U'' si U contient (un cycle contenant) une arête grise dont le support contient la portée de U' mais n'intersecte pas la portée de U''.*

Un exemple illustrant cette définition est donné à la figure 4.3 (source : [10]).

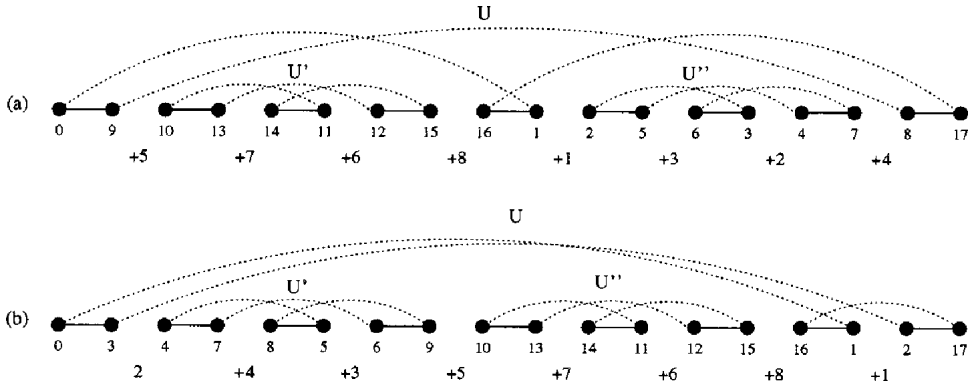


FIG. 4.3 – (a) La composante U sépare les composantes U' et U'' ; (b) U ne sépare pas U' et U'' (source : [10])

Définition 4.15 *Soit $\prec$ un ordre partiel sur un ensemble P ; un élément $p \in P$ est* minimal *s'il n'existe pas de q dans P tel que $q \prec p$, et* maximum *si pour tout $q \in P : q \prec p$.*

Nous avons maintenant tous les outils nécessaires pour présenter la notion de *haie*.

Définition 4.16 *Soit $\mathcal{U}_\pi$ l'ensemble des composantes paires du graphe H_π des chevauchements de cycles ; munissons cet ensemble de l'ordre partiel $\prec$ défini comme suit :*

$$\forall\, U, V \in \mathcal{U}_\pi : U \prec V \Leftrightarrow [U_{min} \overset{\pi'}{---} U_{max}] \subset [V_{min} \overset{\pi'}{---} V_{max}]$$

Tout élément minimal de $\mathcal{U}_\pi$ dans $\prec$ est une haie *; de plus, l'élément maximum de $\mathcal{U}_\pi$ est une haie si et seulement s'il ne sépare pas deux éléments minimaux.*

Sur la figure 4.3, U' et U'' sont des haies, et la composante maximale U n'est une haie que sur la figure 4.3 (b).

Les auteurs des différents articles sur lesquels je me suis basé utilisent invariablement le terme d'"intervalle" pour désigner plusieurs notions différentes ; afin de clarifier la situation, je propose le nouveau terme de *section*, expliqué dans la définition suivante.

Définition 4.17 *Pour une permutation π de n éléments et deux éléments i et j de π (avec $1 \leq |i| \leq |j| \leq n$), la* section $\{i \overset{\pi}{—} j\}$ *est le sous-vecteur*

$$(i\ \pi_{k+1}\ \pi_{k+2}\ \cdots\ \pi_{k+m-1}\ j)$$

du vecteur π où $i = \pi_k$ et $j = \pi_{k+m}$.

Cette définition ressemble fort à la définition 3.1, mais elles ne correspondent pas à la même chose (même si à tout intervalle correspond une section et réciproquement) ; en effet, si l'on reprend l'exemple de la permutation $\pi = (3\ 6\ 7\ 5\ 4\ 1\ 2)$, l'intervalle $[3 \overset{\pi}{—} 4]$ correspond au sous-vecteur $(\pi_3\ \pi_4) = (7\ 5)$, alors que la section $\{3 \overset{\pi}{—} 4\}$ correspond au sous-vecteur $(3\ 6\ 7\ 5\ 4)$. Un intervalle est donc indicé par des positions et une section par des éléments. Nous adapterons à cette notation les notations évidentes déjà utilisées pour les intervalles (semi-) ouverts et fermés.

Deux opérations permettent de se débarrasser des haies :

1. la *découpe*, qui consiste à inverser la section $]i \overset{\pi}{—} i+1[$ d'une haie ;

$$\begin{array}{c} i\ \underline{\pi_{j+1}\ \pi_{j+2}\ \cdots}\ i+1\ \cdots\ \pi_{j+k-2}\ \pi_{j+k-1}\ i+k \\ \Downarrow \\ i\ \cdots\ -\pi_{j+2}\ -\pi_{j+1}\ i+1\ \cdots\ \pi_{j+k-2}\ \pi_{j+k-1}\ i+k \end{array}$$

2. la *fusion* de deux haies, qui consiste à inverser la section $\{i+k \overset{\pi}{—} j\}$, où $i+k$ est le dernier élément de la première haie et j le premier élément de la seconde haie ; nous avons donc :

$$\begin{array}{c} \underbrace{i\ \cdots\ \pi_{t-2}\ i+k}_{\text{haie 1}}\ \pi_t\ \pi_{t+1}\ \cdots\ \pi_u\ \underbrace{j\ \pi_{u+2}\ \cdots\ j+l}_{\text{haie 2}} \\ \Downarrow \\ i\ \cdots\ \pi_{t-2}\ -j\ -\pi_u\ \cdots\ -\pi_{t+1}\ -\pi_t\ -(i+k)\ \pi_{u+2}\ \cdots\ j+l \end{array}$$

Il n'est pas approprié d'appliquer ces opérations dans tous les cas, car la suppression d'une haie peut accidentellement en créer une nouvelle ; par exemple, la découpe de la haie U' dans la permutation de la figure 4.3 (a) élimine U' mais transforme la composante maximale en haie, puisque celle-ci ne sépare plus deux éléments minimaux.

Définition 4.18 *Une haie $K \in \mathcal{U}_\pi$ est une* super haie *si la suppression de K par découpe ou fusion avec une autre haie transforme une composante paire U de H_π qui n'est pas une haie en haie ; dans le cas contraire, K est une* haie simple.

Autrement dit, une haie est simple si son élimination (par découpe ou fusion) réduit le nombre de haies dans la permutation π. Sur la figure 4.4, les composantes A, E et F et sont des haies simples ; la composante C est une super-haie, car son élimination transforme B en haie, comme le montre la figure 4.4 (b).

Définition 4.19 *Une* forteresse *est une permutation possédant un nombre impair de haies qui sont toutes des super haies.*

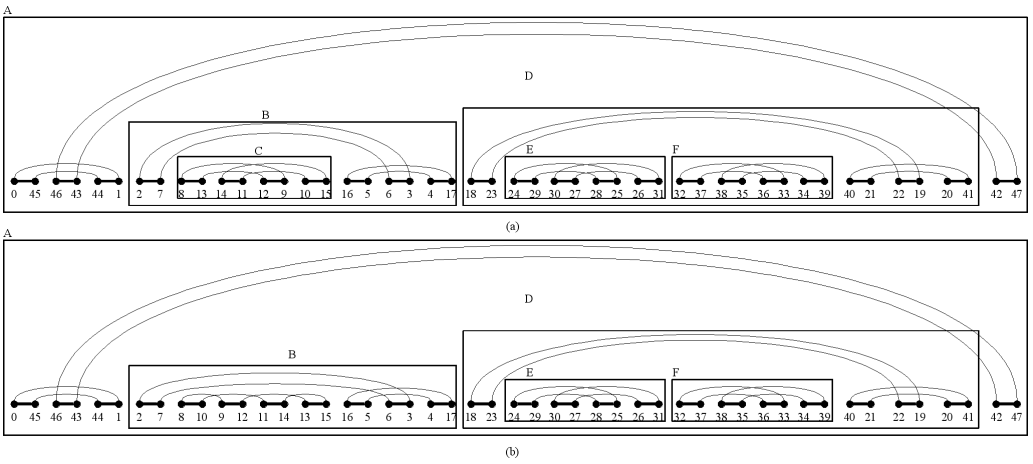


FIG. 4.4 – (a) Une permutation comportant une super haie : C ; (b) la même permutation après la découpe de C : B est maintenant une haie (source : [10])

Hannenhalli et Pevzner [10] ont montré que le nombre minimal d'inversions nécessaires au tri d'une permutation $\pi = (\pi_1\ \pi_2\ \cdots\ \pi_n)$ est

$$d(\pi) = n + 1 - c + h + f \tag{4.1}$$

où :
- c est le nombre de cycles de $G(\pi)$;
- h le nombre de haies de $G(\pi)$;
- f est un facteur de correction valant 1 si π est une forteresse, 0 sinon.

4.4 Calcul des scores

Reprenons le graphe des cycles dont nous avons parlé un peu plus tôt, pour introduire les définitions suivantes.

Définition 4.20 *Le* graphe de chevauchement des arêtes *est le graphe dont les sommets sont les arêtes grises du graphe des cycles et dont les arêtes joignent les sommets représentant les arêtes grises qui se chevauchent dans le graphe des cycles.*

Dans le graphe de chevauchement des arêtes, les sommets impairs seront représentés par un point noir, et les sommets pairs par un point blanc. Attention, ce graphe introduit par Bergeron [2] n'est pas le même que le graphe de chevauchement des *cycles* (définition 4.12) !

Le graphe de chevauchement de la permutation $\pi = (0\ -1\ 3\ 5\ 4\ 6\ -2\ 7)$ est montré à la figure 4.5 (source : [2]).

Fait 4.1 *Un sommet du graphe de chevauchement des arêtes possède un degré impair si et seulement si ce sommet est impair.*

Démonstration :

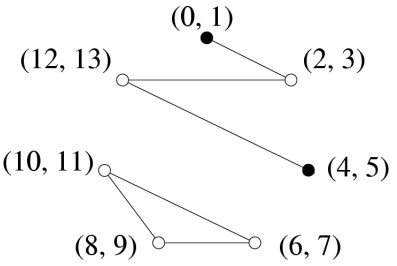


FIG. 4.5 – Graphe de chevauchement des arêtes de $\pi = (0\ {-1}\ 3\ 5\ 4\ 6\ {-2}\ 7)$ (source : [2])

1. $\Leftarrow$: considérons le couple orienté (π_i, π_j) ; les configurations possibles pour ce couple sont :
   (a) $(\pi_i, \pi_j) = (x, -(x+1))$;
   (b) $(\pi_i, \pi_j) = (-x, x+1)$;
   (c) $(\pi_i, \pi_j) = (x+1, -x)$;
   (d) $(\pi_i, \pi_j) = (-(x+1), x)$.

   Dans tous les cas, nous obtenons dans π' les quatre mêmes entiers non signés correspondants $2x-1, 2x, 2x+1$ et $2x+2$ (seul leur ordre d'apparition diffèrera selon le cas). Comme par définition, π_i et π_j sont de signe opposé, les indices des positions de $2x$ et $2x+1$ ont même parité ; cela implique que la section $\{2x \overset{\pi'}{—} 2x+1\}$ (ou $\{2x+1 \overset{\pi'}{—} 2x\}$ selon le cas) comporte un nombre impair d'éléments et donc que l'arête associée à ce support est impaire (définition 4.9). De plus, cette arête chevauche un nombre impair d'intervalles : en effet, une arête impaire possède $2k+1$ éléments, avec $k \geq 1$. Sachant que dans le graphe des cycles, tout sommet est relié exactement à deux autres sommets :
   – soit on relie deux sommets appartenant à la section ouverte considérée, et alors l'arête ainsi créée ne chevauchera pas celle qui joint $2x$ et $2x+1$, et le nombre de sommets à joindre diminue alors de 2 ;
   – soit on relie deux sommets dont l'un n'appartient pas à la section ouverte considérée, et alors l'arête ainsi créée chevauchera celle qui joint $2x$ et $2x+1$, et le nombre de sommets à joindre diminuera de 1.

   Comme la section considérée comporte un nombre impair d'éléments, en procédant de la manière décrite ci-dessus, nous arriverons bien à un nombre impair d'arêtes chevauchant celle qui est associée à la section considérée ; le sommet représentant cette arête dans le graphe de chevauchement aura donc bien un degré impair.

2. $\Rightarrow$: un sommet de degré impair correspond par définition à une arête chevauchant un nombre impair d'arêtes, donc (en se basant sur les supports des arêtes) à un intervalle chevauchant un nombre impair d'intervalles. Or, un tel intervalle doit être de longueur impaire, donc les positions de ses bornes sont de même parité, ce qui implique que le couple correspondant d'entiers consécutifs est orienté. □

Dans les deux faits suivants, l'*inversion correspondant à un sommet impair* désigne l'inversion associée au couple orienté correspondant dans π.

Fait 4.2 *L'inversion associée à un sommet impair v du graphe de chevauchement des arêtes complémente le sous-graphe du graphe de chevauchement constitué de v et de ses sommets adjacents.*

Démonstration : dans le graphe des cycles, l'inversion correspondant à un sommet impair v compresse l'intervalle associé (le support de l'arête), donc v devient isolé. Si u et w sont deux intervalles chevauchant v, exactement un de leurs sommets appartient à l'intervalle associé à v. L'inversion associée à v inversera ces deux points, comme le montre la figure 4.6 (source : [2]). □

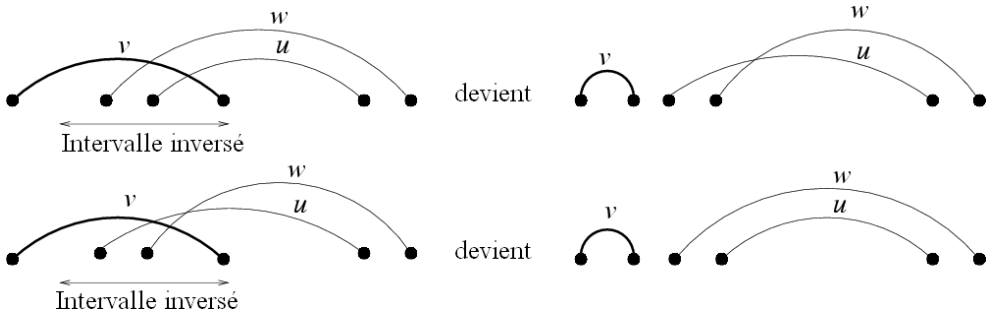


FIG. 4.6 – Effet d'une inversion sur le graphe des cycles (source : [2])

Illustrons (figure 4.7) ce fait par l'application de l'inversion orientée associée au sommet (4,5) dans notre exemple.

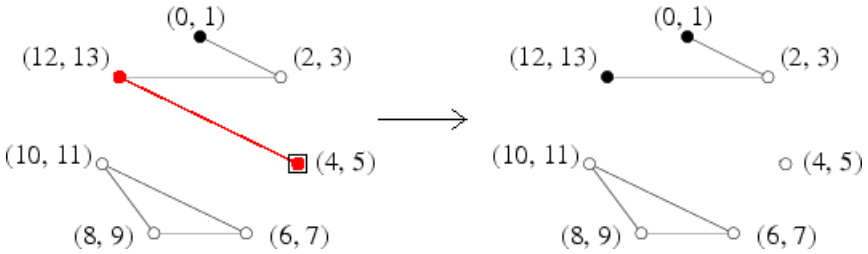


FIG. 4.7 – Illustration du fait 4.2

Fait 4.3 *L'application de l'inversion correspondant à un sommet impair v provoque le changement d'orientation de ses sommets adjacents.*

Démonstration : comme v est impair, il possède un nombre impair $2k+1$ ($k \geq 1$) de sommets adjacents ; soit w l'un de ces sommets, possédant lui-même j voisins également adjacents à v. L'inversion fera perdre $j+1$ voisins à w, et lui en fera gagner $2k-j$; son degré changera donc de $2k-2j-1$, ce qui changera son orientation. □

Fait 4.4 *Le score de l'inversion impaire correspondant au sommet impair v est donné par $T + U - O - 1$, où :*

– *T est le nombre de sommets impairs du graphe ;*
– *U est le nombre de sommets pairs du graphe adjacents à v ;*
– *O est le nombre de sommets impairs du graphe adjacents à v.*

Démonstration : découle des faits précédents. □

Définition 4.21 *Une composante connexe du graphe de chevauchement des arêtes est* orientée *si elle contient au moins un sommet impair,* non orientée *sinon.*

Définition 4.22 *Une inversion est* sûre *si elle ne crée pas de nouvelles composantes non orientées (sauf pour les sommets isolés) dans le graphe de chevauchement des arêtes.*

Théorème 4.1 *[10] Toute séquence d'inversions orientées sûres est optimale.*

Ainsi, la difficulté du tri de composantes orientées réside dans la détection d'inversions sûres.

Théorème 4.2 *Toute inversion orientée de score maximal est sûre.*

Démonstration : procédons par l'absurde : supposons que l'inversion soit de score maximal, mais non sûre ; en d'autres termes, soit un sommet v de score maximal, et supposons que l'inversion associée crée une nouvelle composante paire C contenant au moins deux sommets. Alors, au moins l'un des sommets de C devait être adjacent à v avant l'inversion, puisque les seules arêtes affectées par l'inversion sont celles situées entre les sommets adjacents à v. Soit w un tel sommet ; nous avons :

$$\begin{aligned} score(v) &= T + U - O - 1 \\ score(w) &= T' + U' - O' - 1 \end{aligned}$$

Tous les sommets pairs qui étaient adjacents à v devaient être adjacents à w ; en effet, un sommet pair adjacent à v et pas à w deviendra impair et connecté à w, contrairement à l'hypothèse selon laquelle C est paire. Donc $U' \geq U$.

Tous les sommets impairs qui étaient adjacents à w devaient être adjacents à v ; si ce n'était pas le cas, un sommet impair adjacent à w mais pas à v resterait impair, contrairement à l'hypothèse selon laquelle C est paire. Donc $O' \leq O$.

Si $U' = U$ et $O' = O$, v et w ont le même ensemble de sommets adjacents et le complément du sous-graphe de v et de ses sommets adjacents isole v et w. Donc $score(w) > score(v)$, ce qui est impossible car v est de score maximal. □

4.5 Algorithmes

On peut considérer qu'il existe deux classes d'algorithmes liés au problème MIN-SBR signé : les algorithmes calculant $d(\pi)$, et ceux qui fournissent en plus la séquence

d'inversions à appliquer à π. Les progrès qui ont été faits dans l'étude de ce problème sont suffisamment remarquables pour être soulignés : en effet, ce problème était supposé NP-difficile, à l'instar de son homologue non signé, jusqu'à ce que Hannenhalli et Pevzner [10] fournissent deux algorithmes de complexités respectives $O(n^5)$ et $O(n^4)$; des algorithmes linéaires ont récemment été conçus pour calculer la distance $d(\pi)$, et les meilleurs algorithmes pour la recherche d'une séquence optimale d'inversions étaient jusqu'à très récemment en $O(n^2)$, ce qui a amené la question de savoir s'il était possible de trouver un algorithme de complexité sous-quadratique pour trouver une séquence d'inversions optimale. Tannier [21] a apporté une réponse positive à cette question, et nous parlerons de son algorithme en $O(n\sqrt{n \log n})$ dans la section 4.5.1.

4.5.1 Recherche d'une séquence d'inversions optimale

Algorithme(s) de Bergeron [2]

Cet exemple a surtout été retenu pour des raisons pédagogiques car il est simple à énoncer, à comprendre et permet d'illustrer les notions précédentes et de mieux mettre en valeur les avantages des algorithmes plus performants qui ont été ensuite créés.

Algorithme 4.1 *Tant que la permutation π possède un couple orienté, choisir et appliquer une inversion orientée de score maximal.*

L'optimalité de l'algorithme 4.1 est justifiée par le théorème 4.2. La figure 4.8 présente un même exemple d'application de l'algorithme 4.1, avec le traitement de quelques ambiguïtés (donc les cas où il existe plusieurs inversions orientées de score maximal) ; les flèches noires représentent les choix effectués dans [2], les flèches rouges sont des choix alternatifs et la notation $(x, y)_s$ désigne le choix de l'inversion orientée de score s associée au couple (x, y). Remarquons qu'il arrive que deux inversions différentes agissent sur le même intervalle.

Cet exemple montre bien que la solution du problème n'est pas unique : plusieurs séquences minimales d'inversions mènent à la solution et c'est pourquoi le problème MIN-SBR spécifie que l'on recherche *une* telle séquence. Comme l'exprime le théorème 4.2, n'importe quelle inversion orientée de score maximal peut être sélectionnée à chaque étape.

L'algorithme 4.1 suppose que π possède des couples orientés ; si ce n'est pas le cas, donc si π est positive, il va falloir en créer, et cela va se faire par l'élimination des haies de la permutation.

Définition 4.23 *Un* intervalle encadré *dans une permutation π est une section $\{i \overset{\pi}{—} i+k\}$ dont tous les éléments appartiennent à l'intervalle entier*[1] $[i, i+k]$.

Remarquons qu'une permutation encadrée forme elle-même un intervalle encadré ; un intervalle encadré peut être vu comme un morceau de ι, dont les éléments sont éventuellement mélangés. Bergeron [2] considère des permutations positives,

[1] A ne pas confondre avec la définition 3.1.

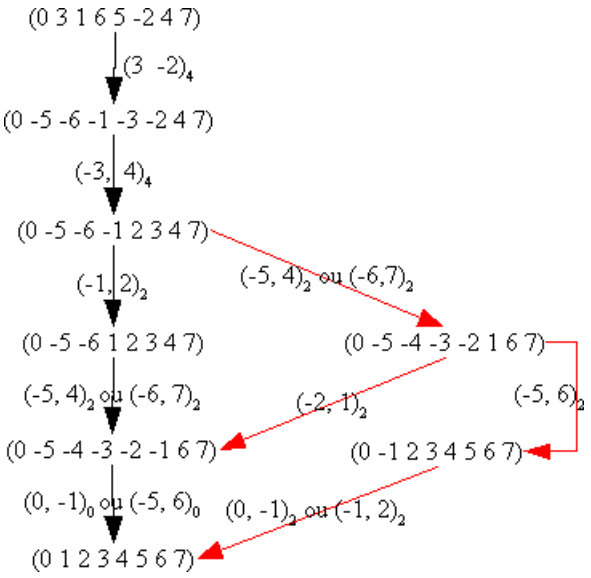


FIG. 4.8 – Application de l'algorithme 4.1 à (0 3 1 6 5 −2 4 7)

encadrées par 0 et $n+1$, et les munit d'un ordre circulaire (0 est le successeur de $n+1$) ; dans ce contexte, elle redéfinit une haie comme un intervalle encadré n'en contenant aucun et ne se soucie pas du cas signé, car s'il y a des éléments négatifs, il suffit d'appliquer l'algorithme 4.1.

Algorithme 4.2 *Si la permutation π comporte $2k$ haies ($k \geq 2$) : fusionner toutes les haies non consécutives. Si π comporte $2k+1$ haies ($k \geq 1$) : si π contient une haie simple, la découper, sinon fusionner deux haies non consécutives ou consécutives si $k = 1$.*

Cet algorithme, en conjonction avec l'algorithme 4.1, permet de trier de manière optimale une permutation donnée. Concluons en précisant que cet algorithme est de complexité $O(n^3)$ (les détails s'appuient sur la dernière section de [2], qui consiste en l'exposé de la version "parallèle" de l'algorithme).

Algorithme de Tannier [21]

Nous allons maintenant présenter un algorithme permettant de trouver en $O(n\sqrt{n \log n})$ une séquence optimale d'inversions triant une permutation donnée ; cet algorithme a été inventé par Tannier et ne s'applique qu'aux permutations simples (définition 4.25) et sans haie. Ceci n'est toutefois pas restrictif, car il existe des algorithmes linéaires [13] pour transformer toute permutation en une telle permutation.

Définition 4.24 *La* taille *d'un cycle est le nombre d'arêtes grises qu'il contient.*

Définition 4.25 *Une permutation π est* simple *si les cycles de son graphe des points de rupture sont de taille 1 et 2, notés respectivement 1-cycles et 2-cycles.*

Remarquons qu'un 1-cycle est en fait une adjacence (définition 3.4). Un exemple de permutation simple est montré à la figure 4.9.

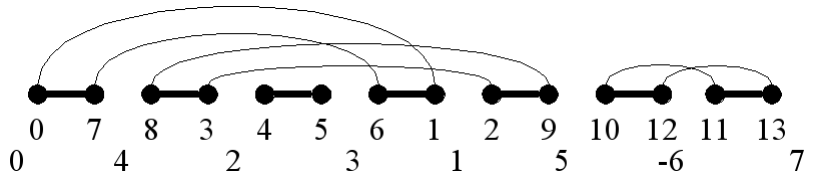


FIG. 4.9 – Une permutation simple

Définition 4.26 *Etant donné le graphe des cycles $G(\pi)$ d'une permutation signée π de n éléments, le* graphe inverse $G^{-1}(\pi)$ *est obtenu en appliquant les transformations suivantes à $G(\pi)$:*

1. *chaque sommet de π' est envoyé sur sa position ;*
2. *toute arête grise devient noire et toute arête noire devient grise.*

Le graphe inverse du graphe des cycles de la permutation $\pi = (0\ -1\ 3\ 5\ 4\ 6\ -2\ 7)$ (figure 4.1) est montré en exemple à la figure 4.10.

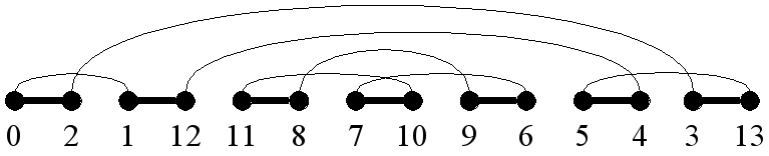


FIG. 4.10 – Graphe inverse du graphe des cycles de la figure 4.1

Lemme 4.1 *Pour toute permutation π : $G^{-1}(\pi) = G(\pi^{-1})$.*

Démonstration : par définition, $G^{-1}(\pi)$ et $G(\pi^{-1})$ ont le même ensemble de sommets. Chaque arête noire $\{\pi'_{2i}, \pi'_{2i+1}\}$ de $G(\pi)$ devient une arête grise dans $G^{-1}(\pi)$ joignant $2i$ et $2i+1$, qui est par définition une arête grise du graphe des cycles de π'^{-1}. De plus, toute arête grise $\{2i, 2i+1\}$ de $G(\pi)$ devient une arête noire dans $G^{-1}(\pi)$ joignant π'^{-1}_{2i} et π'^{-1}_{2i+1}, donc $G^{-1}(\pi)$ et $G(\pi^{-1})$ ont le même ensemble d'arêtes. □

Remarquons qu'une conséquence de ce lemme est qu'à chaque cycle de π correspond un cycle de π^{-1}. La figure 4.11, extraite de [21] (Tannier préfère disposer ce graphe de manière circulaire, mais cela ne doit pas troubler le lecteur : il s'agit bien du même graphe), donne une illustration de ce lemme.

Lemme 4.2 *Pour toute inversion ρ, il vient $\rho = \rho^{-1}$. De plus, si ρ_1, ..., ρ_k sont des inversions, π une permutation et $(\rho_k \circ \cdots \circ \rho_1)\pi = \iota$, alors $\pi^{-1} = (\rho_k \circ \cdots \circ \rho_1)\iota$ et $(\rho_1 \circ \cdots \circ \rho_k)\pi^{-1} = \iota$.*

La démonstration du lemme 4.2 devient aisée lorsque l'on a compris que l'action d'une inversion ρ sur une permutation π revient à "multiplier" π à droite par une certaine permutation signée (voir à ce sujet la section 6.3).

Lemme 4.3 *[10] Pour toute permutation simple sans haie π, il vient : $d(\pi) = c(\pi)$.*

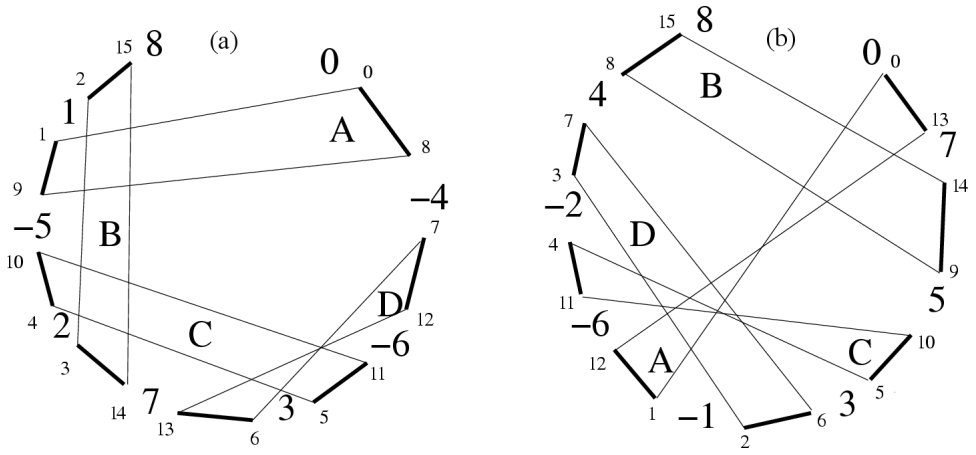


FIG. 4.11 – (a) Le graphe des cycles de $(0\ -4\ -6\ 3\ 7\ 2\ -5\ 1\ 8)$; (b) son graphe inverse (source : [21])

Il faut donc trouver une séquence d'inversions faisant décroître à chaque étape le nombre de cycles, et ceci sans créer de haie.

Lemme 4.4 *Pour tout 2-cycle d'une permutation simple sans haie, ses deux arêtes noires ont même parité et ses deux arêtes grises ont même parité.*

Démonstration : procédons par l'absurde : soit e et f les deux arêtes grises d'un 2-cycle, de parités différentes ; supposons e paire (donc les indices de ses extrémités sont de parités différentes), et f impaire (donc les indices de ses extrémités sont de même parité). Alors le cycle contient par exemple[2] trois sommets d'indices pairs et un sommet d'indice impair, ce qui est impossible car les deux arêtes noires du cycle doivent par définition joindre deux couples distincts de sommets de parité différente. Tous les autres cas se démontrent de manière similaire. □

Définition 4.27 *Un cycle est* amovible *si ses arêtes noires sont impaires.*

Par exemple, les cycles A et C du graphe de la figure 4.11 (a) sont amovibles.

Définition 4.28 *Un cycle est* contractile *si ses arêtes grises sont impaires.*

Un exemple de cycle contractile est le cycle C du graphe des cycles de la figure 4.11 (a). Il découle de ces définitions et du lemme 4.1 qu'un cycle est amovible dans π si et seulement s'il est contractile dans π^{-1}.

Remarquons qu'un cycle peut très bien n'être ni amovible ni contractile : c'est le cas par exemple du cycle B de la figure 4.11 (a).

Lemme 4.5 *[10] Un cycle C_ρ est contractile si et seulement si $c(\rho(\pi)) = c(\pi) - 1$.*

Attention, cela ne signifie pas que $d(\rho(\pi)) = d(\pi) - 1$, car $\rho(\pi)$ peut contenir des haies. Les lemmes 4.1, 4.2 et 4.5 permettent de déduire le lemme suivant.

[2] Si deux extrémités de e et f ne coïncident pas.

Lemme 4.6 *Un cycle C_ρ est amovible si et seulement si $c(\rho(\iota) \circ \pi) = c(\pi) - 1$.*

Définition 4.29 *Soit C_ρ un 2-cycle de $G(\pi)$, contenant les arêtes noires $\{\pi'_{2i}, \pi'_{2i+1}\}$ et $\{\pi'_{2j}, \pi'_{2j+1}\}$ $(i < j)$; la* contraction *de ce cycle consiste à :*

1. *remplacer ses arêtes grises par deux nouvelles arêtes grises parallèles à ses arêtes noires ;*
2. *$\forall\ i+1 \leq r \leq j$: échanger les sommets $2r-1$ et $2r$.*

Le graphe résultant de cette contraction est noté $G(\pi)/C_\rho$.

Reprenons par exemple le graphe de la figure 4.11 : la contraction du cycle C aboutit au graphe montré à la figure 4.12 (source : [21]).

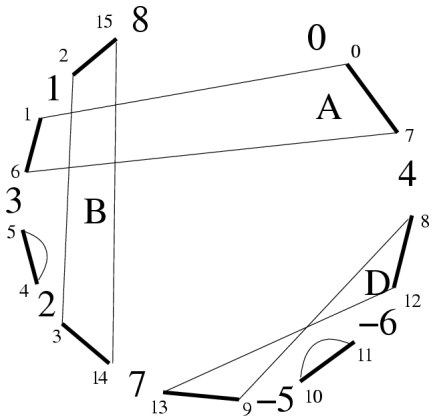


FIG. 4.12 – Le graphe de la figure 4.11 (a) après contraction du cycle C (source : [21])

Lemme 4.7 *[10] Si un cycle C_ρ de $G(\pi)$ est contractile, alors $G(\pi)/C_\rho = G(\rho(\pi))$.*

Définition 4.30 *Soit C_ρ un 2-cycle de $G(\pi)$, contenant les arêtes noires $\{\pi'_{2i}, \pi'_{2i+1}\}$ et $\{\pi'_{2j}, \pi'_{2j+1}\}$ $(i < j)$; l'*élimination *de ce cycle consiste à :*

1. *remplacer ses arêtes grises par deux nouvelles arêtes grises parallèles à ses arêtes noires ;*
2. *modifier de façon adéquate les sommets affectés par cette transformation.*

Le graphe résultant de cette contraction est noté $G(\pi) \setminus C_\rho$, qui est le graphe inverse de $G^{-1}(\pi)/C_\rho$.

Le graphe résultant de l'élimination du cycle C dans le graphe de la figure 4.11 (b) est montré en exemple à la figure 4.13.

Lemme 4.8 *Si un cycle C_ρ de $G(\pi)$ est amovible, alors $G(\pi) \setminus C_\rho = G(\rho(\iota) \circ \pi)$.*

Lemme 4.9 *Soit C_ρ, $C_{\rho'}$ deux cycles distincts de π ; alors :*

1. *si C_ρ est contractile, alors $C_{\rho'}$ est amovible dans π si et seulement si $C_{\rho'}$ est amovible dans $\rho(\pi)$;*

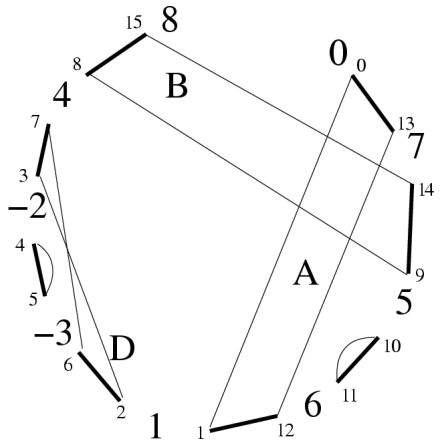


FIG. 4.13 – Le graphe de la figure 4.11 (b) après élimination du cycle C (source : [21])

2. *si C_ρ est amovible, alors $C_{\rho'}$ est amovible dans π si et seulement si $C_{\rho'}$ est contractile dans $\rho(\iota) \circ \pi$;*
3. *si C_ρ est amovible, alors deux arêtes distinctes de celles de C_ρ se chevauchent dans $G(\pi)$ si et seulement si elles se chevauchent dans $G(\rho(\pi))$.*

Démonstration :

1. soit $\{\pi'_{2r}, \pi'_{2r+1}\}$ une arête noire de $C_{\rho'}$; selon que $2r$ appartient ou non à l'intervalle inversé par ρ, les positions de ses extrémités changeront ou non de parité : mais dans les deux cas, l'orientation ne sera pas modifiée, donc $C_{\rho'}$ est amovible dans π si et seulement s'il est amovible dans $\rho(\pi)$;
2. se déduit des lemmes 4.1 et 4.8 ;
3. soit $\{2i, 2i{+}1\}$ et $\{2j, 2j{+}1\}$ (nous supposons $|\pi'^{-1}_{2i}| < |\pi'^{-1}_{2j}|$) deux arêtes grises de $G(\pi)$, et $\{2i', 2i'{+}1\}$ et $\{2j', 2j'{+}1\}$ les mêmes arêtes dans $G(\rho(\iota)\circ\pi)$ (donc après l'élimination de C_ρ). Les arêtes grises $\{2i, 2i+1\}$ et $\{2i', 2i'+1\}$ correspondent respectivement aux arêtes noires $\{\pi'^{-1}_{2i}, \pi'^{-1}_{2i+1}\}$ et $\{(\rho(\iota)\circ\pi^{-1})'_{2i'}, (\rho(\iota)\circ \pi^{-1})'_{2i'+1}\} = \{(\rho(\pi^{-1}))'_{2i'}, (\rho(\pi^{-1}))'_{2i'+1}\}$. Toutefois, la contraction d'un cycle ne modifie pas ces sommets (lemme 4.7), donc $|(\rho(\pi^{-1}))'_{2i'}| = |\pi'^{-1}_{2i}|$ et $|(\rho(\pi^{-1}))'_{2i'+1}| = |\pi'^{-1}_{2i+1}|$ (et cela est valable également pour j). Cela signifie que $|\pi'^{-1}_{2i}| < |\pi'^{-1}_{2j}| < |\pi'^{-1}_{2i+1}| < |\pi'^{-1}_{2j+1}|$ si et seulement si $|(\rho(\iota) \circ \pi^{-1})'_{2i'}| < |(\rho(\iota) \circ \pi^{-1})'_{2j'}| < |(\rho(\iota) \circ \pi^{-1})'_{2i'+1}| < |(\rho(\iota) \circ \pi^{-1})'_{2j'+1}|$, ce qui prouve que les arêtes grises se chevauchent dans $G(\rho(\iota) \circ \pi)$ si et seulement si elles se chevauchent dans $G(\pi)$. □

Reprenons les composantes dont nous avons parlé en section 4.3 ; chacune des composantes impaires du graphe de chevauchement des cycles pouvant être triée séparément, il sera supposé sans perte de généralité qu'il n'existe qu'une seule telle composante.

Définition 4.31 *Une* séquence valide *d'une permutation π est un ordonnancement d'un sous-ensemble de ses 2-cycles C_{ρ_1}, ..., C_{ρ_k} tel que $\forall\ i \in \{1, ..., k\}$, C_{ρ_i} est un*

cycle amovible de $(\rho_{i-1} \circ \cdots \circ \rho_1)(\iota) \circ \pi$. *En d'autres termes,* C_{ρ_1}, ..., C_{ρ_k} *est valide si* $c((\rho_k \circ \cdots \circ \rho_1)(\iota) \circ \pi) = c(\pi) - k$.

Définition 4.32 *Une séquence valide d'une permutation* π *est* maximale *si aucun cycle de* $(\rho_k \circ \cdots \circ \rho_1)(\iota) \circ \pi$ *n'est amovible et* totale *si* $k = c(\pi)$.

L'algorithme de Tannier procède en deux étapes :

1. calculer une séquence valide maximale de π ;
2. augmenter la taille de cette séquence en lui ajoutant des cycles tant qu'elle n'est pas totale.

La première étape consiste à détecter les cycles amovibles et à les éliminer tant qu'il en existe dans le graphe résultant de la transformation qui vient d'être appliquée et ceci est facilement faisable en temps linéaire. Le Théorème 4.3 prouve que la seconde étape de l'algorithme est toujours réalisable.

Théorème 4.3 *Si* S *est une séquence valide maximale mais non totale pour une permutation* π, *alors il existe une séquence non vide* S' *telle que* S *peut être partitionnée en deux sous-séquences* S_1, S_2 *de telle sorte que* S_1, S', S_2 *est une séquence valide pour* π.

Démonstration : soit $S = C_{\rho_1}, ..., C_{\rho_k}$ une séquence valide maximale de π ; l'ensemble $\mathcal{C}$ des cycles de la permutation $(\rho_k \circ \cdots \circ \rho_1)(\iota) \circ \pi)$ est l'union de composantes paires (sinon il subsisterait un cycle amovible et S ne serait pas maximale). Comme tous les cycles de π se chevauchent, il existe nécessairement un cycle de S qui chevauche un cycle de $\mathcal{C}$; choisissons un tel cycle $C_{\rho_l} \in S$ tel que l est maximal, et définissons $S_1 = C_{\rho_1}, ..., C_{\rho_{l-1}}$ et $S_2 = C_{\rho_l}, ..., C_{\rho_k}$. Enfin, notons $\sigma = (\rho_{l-1} \circ \cdots \circ \rho_1)(\iota) \circ \pi$ la permutation obtenue à partir de π en retirant de cette dernière les cycles de S_1 ; les lemmes suivants vont permettre de prouver ce théorème.

Lemme 4.10 *Au moins un cycle de chaque composante de* $\mathcal{C}$ *chevauchée par* ρ_l *est amovible dans* σ.

Démonstration : comme une inversion ne modifie pas la parité des arêtes grises (lemme 4.9), et que $\mathcal{C}$ est paire dans $\sigma = (\rho_k \circ \cdots \circ \rho_1)(\iota) \circ \pi$, aucun cycle de $\mathcal{C}$ n'est contractile dans π ni dans σ. Soit C_ρ un cycle de $\mathcal{C}$ chevauchant C_{ρ_l} dans σ, et $\{\sigma'_{2i}, \sigma'_{2i+1}\}$ une arête noire de C_{ρ_l} ; comme C_{ρ_l} est amovible, cette arête est impaire et σ_i et σ_{i+1} ont le même signe. Soit $\{2j, 2j+1\}$ une arête grise de C_ρ chevauchant C_{ρ_l} ; nous avons $2j < 2i$ et $2j' > 2i+1$. Comme $\{\sigma'_{2j}, \sigma'_{2j'}\}$ est paire, il y a donc un nombre pair d'arêtes noires impaires parmi celles situées dans l'intervalle couvert par $\{\sigma'_{2j}, \sigma'_{2j'}\}$, donc au moins un cycle amovible distinct de C_{ρ_l} doit chevaucher C_ρ, et comme l est maximal, ce cycle est dans $\mathcal{C}$. □

Soit $S' = C'_{\rho_1}, ..., C'_{\rho_p}$ une séquence valide de cycles de $\mathcal{C}$ dans σ, telle que $\mathcal{C} \setminus S'$ ne contient aucun cycle amovible dans $(\rho'_p \circ \cdots \circ \rho'_1)(\iota) \circ \sigma$.

Lemme 4.11 *Le cycle* ρ_l *est amovible dans* $(\rho'_p \circ \cdots \circ \rho'_1)(\iota) \circ \sigma$.

Démonstration : soit $\tau = (\rho'_{p-1} \circ \cdots \circ \rho'_1)(\iota) \circ \sigma$, et $\{\tau'_{2i}, \tau'_{2i+1}\}$, $\{\tau'_{2j}, \tau'_{2j+1}\}$ les deux arêtes noires de C'_{ρ_p} dans τ. Soit

$$M = \max\{\tau'_{2i}, \tau'_{2i+1}, \tau'_{2j}, \tau'_{2j+1}\} \text{ et } m = \min\{\tau'_{2i}, \tau'_{2i+1}, \tau'_{2j}, \tau'_{2j+1}\}$$

Comme C'_{ρ_p} est contractile dans τ, τ'_{2i} et τ'_{2i+1} sont de signes opposés, et il en va de même pour τ'_{2j} et τ'_{2j+1} ; comme $\tau'_{2j+1} = \tau'_{2i} + 1$ et $\tau'_{2i+1} = \tau'_{2j} + 1$, M et m ne peuvent être adjacents et et les éléments leur correspondant dans τ sont de signes opposés. Eliminons C'_{ρ_p} de τ, donc remplaçons les arêtes grises par deux nouvelles arêtes grises parallèles aux arêtes noires du cycle et modifions les sommets compris entre m et M (ces derniers non compris) (lemme 4.8). Dans le graphe résultant de cette transformation, m et M appartiennent à des adjacences et sont "de signes opposés" ; il y a donc un nombre impair de changement de signes entre m et M, et donc un nombre impair d'arêtes noires impaires. Comme les arêtes noires impaires appartiennent à un 2-cycle amovible (elles se rencontrent uniquement par paires), au moins l'un de ces 2-cycles chevauchent C'_{ρ_p} dans τ. Comme il n'y a par hypothèse aucun cycle amovible de $\mathcal{C}$ dans $\rho'_p(\iota) \circ \tau$ et que C_{ρ_l} est le seul cycle n'appartenant pas à $\mathcal{C}$ qui puissent chevaucher un cycle de $\mathcal{C}$, ce cycle éliminable est C_{ρ_l}. □

Lemme 4.12 *La séquence S_1, S', S_2 est une séquence valide pour π.*

Démonstration : par le lemme précédent, nous savons déjà que C_{ρ_1}, ..., $C_{\rho_{l-1}}$, C'_{ρ_1}, ..., C'_{ρ_p}, C_{ρ_l} est une séquence valide. Prenons maintenant $\sigma = (\rho_l \circ \cdots \circ \rho_1)(\iota) \circ \pi$, et $\tau = (\rho_l \circ \rho'_p \circ \cdots \circ \rho'_1 \circ \rho_{l-1} \circ \cdots \circ \rho_1)\pi$. Soit D_1 la composante de $G(\sigma)$ induite par les cycles $C_{\rho_{l+1}}$, ..., C_{ρ_k} dans σ ; comme chaque composante du graphe des cycles peut être triée séparément, nous pouvons dire que $C_{\rho_{l+1}}$, ..., C_{ρ_k} est une séquence valide totale pour D_1. Soit D_2 la composante de $G(\tau)$ induite par les cycles $C_{\rho_{l+1}}$, ..., C_{ρ_k} dans τ ; un cycle C_{ρ_i} est contractile dans D_1 si et seulement s'il est contractile dans D_2, et deux cycles se chevauchent dans D_1 si et seulement s'ils se chevauchent dans D_2 (ils ne diffèrent que par quelques inversions). Alors $C_{\rho_{l+1}}$, ..., C_{ρ_k} est une séquence valide totale pour D_2 ; enfin, C_{ρ_1}, ..., $C_{\rho_{l-1}}$, $C_{\rho'_1}$, ..., C'_{ρ_p}, C_{ρ_l}, ..., C_{ρ_k} est valide pour π. □

Ceci conclut également la preuve du Théorème 4.3. □

L'algorithme décrit plus haut serait de complexité quadratique sans le recours à la structure de données de Kaplan et Verbin [14] ; celle-ci permet en effet de choisir et d'appliquer une inversion ρ associée à un cycle contractile C_ρ en un temps sous-linéaire. Elle va être utilisée pour représenter π^{-1} (puisque nous cherchons des cycles amovibles dans π, donc contractiles dans π^{-1}) de la façon suivante : la permutation sera découpée en blocs de taille $O(\sqrt{n \log n})$, et à chacun de ces blocs sera associé un booléen vrai si le bloc doit être lu en sens inverse (et en changeant le signe des éléments). A chaque bloc est également associé un arbre binaire de recherche équilibré (par exemple un *splay tree*[3]), dans les nœuds duquel les éléments du bloc

[3]Il s'agit d'un arbre binaire de recherche équilibré et auto-adaptatif, sur lequel les opérations se font en une complexité amortie $O(\log n)$ grâce à une opération spéciale effectuée à chaque recherche.

sont stockés et triés de façon croissante sur la position de leur successeur dans π^{-1} ; cela signifie que π_i^{-1} est situé avant π_j^{-1} si $\pi_{\pi_i^{-1}+1} < \pi_{\pi_j^{-1}+1}$. Chaque nœud de l'arbre correspond à une arête grise de $G(\pi^{-1})$ (donc chaque 2-cycle est représenté par deux nœuds) et contient également la parité de cette arête, ce qui permet de savoir si le cycle la contenant est contractile et donc amovible dans π, et enfin, le nombre total d'arêtes impaires dans le sous-arbre.

Les ajouts de Tannier à cette structure sont les suivants : soit $\mathcal{C}$ un sous-ensemble des 2-cycles de $G(\pi^{-1})$, initialisé à l'ensemble de tous les 2-cycles. A chaque nœud est ajouté un nouveau booléen, vrai si l'arête correspondante appartient à un cycle de $\mathcal{C}$, et un total contenant le nombre d'arêtes impaires de $\mathcal{C}$ dans le sous-arbre. Ainsi, les opérations décrites dans les lemmes suivants sont réalisables de la même manière et avec la même complexité que chez Kaplan et Verbin [14].

Lemme 4.13 *[14] Il est possible de savoir en temps constant s'il existe un cycle amovible dans $G(\pi)$ et de choisir un tel cycle en temps $O(\log n)$.*

Nous admettons le lemme suivant sans démonstration.

Lemme 4.14 *Il est possible de savoir en temps constant s'il existe un cycle amovible de $G(\pi)$ dans $\mathcal{C}$ et le cas échéant de choisir un tel cycle en temps $O(\log n)$.*

Lemme 4.15 *Il est possible de contracter un cycle et de maintenir la structure de données en temps $O(\sqrt{n \log n})$.*

Démonstration : la seule information à maintenir en plus de ce qui est fait dans [14] est l'appartenance d'une arête à $\mathcal{C}$. Pour ce faire, il suffit d'éliminer une arête de $\mathcal{C}$ quand elle est contractée puis de mettre à jour les totaux. Le nombre d'éléments de $\mathcal{C}$ décroît donc de 1 dans tous les blocs contenant le cycle contracté, et il faut calculer la différence entre le nombre d'éléments de $\mathcal{C}$ et le nombre de cycles contractiles de $\mathcal{C}$ avant l'opération dans tous les blocs. □

Enfin, le Théorème suivant prouve la complexité de l'algorithme de Tannier.

Théorème 4.4 *Il est possible de trouver une séquence totale d'inversions triant π en un temps $O(n\sqrt{n \log n})$.*

Démonstration : la construction de la séquence $S = C_{\rho_1}, ..., C_{\rho_k}$ prend un temps $O(k\sqrt{n \log n})$; si cette séquence n'est pas totale, il subsiste un ensemble $\mathcal{C}$ de cycles à éliminer (si par contre elle est totale, nous avons fini). Il faut donc trouver un cycle $C_{\rho_l} \in S$ tel qu'il existe un cycle amovible dans $\mathcal{C}$ dans $(\rho_{l-1} \circ \cdots \circ \rho_1)(\iota) \circ \pi$ et que l est maximal ; cela peut se faire en "réinsérant" un à un les cycles retirés $C_{\rho_k}, ..., C_{\rho_l}$, en vérifiant à chaque étape en temps constant si $\mathcal{C}$ comporte un cycle amovible et en s'arrêtant dès que c'est le cas. Ceci prend un temps $O((k-l)\sqrt{n \log n})$; la séquence $\rho_k, ..., \rho_l$ est alors une séquence d'inversions sûres dont les cycles correspondants sont contractiles dans π^{-1} et seront les dernières inversions de la séquence finale (ce morceau de séquence n'est donc plus modifié).

Il faut ensuite retirer tous les cycles amovibles de $\mathcal{C}$; cela donne la séquence $\rho_{k+1}, ..., \rho_{k'}$ que l'on insère après la séquence $\rho_1, ..., \rho_{l-1}$. Si la séquence $\rho_1, ..., \rho_{l-1}$,

ρ_{k+1}, ..., $\rho_{k'}$, ρ_k, ..., ρ_l est totale, nous avons fini ; sinon, il faut recommencer en réinsérant les cycles retirés un à un dans l'ordre inverse, mais en commençant par $C_{\rho_{k'}}$ (on ne touche plus à la séquence ρ_k, ..., ρ_l) ; il faut continuer tant qu'il y a un cycle amovible dans $\mathcal{C}$ et recommencer tant que $\mathcal{C}$ n'est pas vide. Tout cycle de la séquence étant retiré une seule fois et remplacé au plus une fois, la complexité totale est $O(n\sqrt{n \log n})$. □

4.5.2 Calcul de $d(\pi)$

La section précédente traitait d'algorithmes fournissant une séquence optimale d'inversions triant une permutation signée π donnée ; nous allons maintenant nous intéresser à un algorithme se bornant à calculer $d(\pi)$ et ne fournissant donc pas une séquence d'inversions triant π.

Un nouvel algorithme pour le calcul de $d(\pi)$ a été développé par Bergeron et Stoye [3] et présente le double avantage de posséder une complexité temporelle et spatiale linéaire et de se passer totalement des structures mentionnées jusqu'ici ; une caractéristique intéressante de cette approche est l'analyse des effets d'une inversion sur une permutation sans réellement appliquer cette inversion, alors qu'un algorithme utilisant le graphe de chevauchement doit appliquer l'inversion (et éventuellement l'annuler si le résultat n'est pas satisfaisant).

Définition 4.33 *Un intervalle* $[i \overset{\pi}{—} j]$ *est* commun *si les éléments* $\{|\pi_i|, ..., |\pi_j|\}$ *sont consécutifs une fois réordonnés.*

Exemple : dans la permutation $(1\ -2\ -4\ 3\ 5)$, l'intervalle $[2 \overset{\pi}{—} 4]$ est commun, car $[2 \overset{\pi}{—} 4] = \{-2, -4, 3\}$ qui, une fois les éléments rendus positifs et triés, consiste en l'ensemble $\{2, 3, 4\}$.

Définition 4.34 *Soit une permutation signée* π*, encadrée par 0 et* n*; à chaque élément* $0 \leq |i| \leq n-1$ *est associée une* inversion élémentaire $r_{|i|}$*, agissant sur la section* $\{i \overset{\pi}{—} i+1\}$ *dont on écarte cependant :*

$$\begin{cases} i & \text{s'il est positif et précède } i+1 \text{ ou s'il est négatif et suit } i+1 \\ i+1 & \text{s'il est négatif et précède } i \text{ ou s'il est positif et suit } i \end{cases}$$

Si les éléments i *et* $i+1$ *sont de signes opposés, l'inversion est dite* orientée *; sinon, elle est* non orientée.

Il est important de comprendre que les notions de "suivre" et "précéder" font référence aux positions des éléments dans la permutation et non pas à un ordre de grandeur ; l'exemple suivant, extrait de [3], illustre bien cette notion :

$$(0\ \underbrace{-6\ 3\ \underbrace{\underbrace{-4}_{r_3=r_4}\ 5}_{r_5}\ \underbrace{2}_{r_1}}_{r_2}\ -1\ 7\ \underbrace{\underbrace{9}_{r_7}\ \underbrace{8}_{r_9}}_{r_8}\ 10)$$
$$r_0=r_6$$

Afin d'alléger les notations, dans la suite, nous noterons r_i l'inversion élémentaire associée à l'élément i (au lieu de $r_{|i|}$).

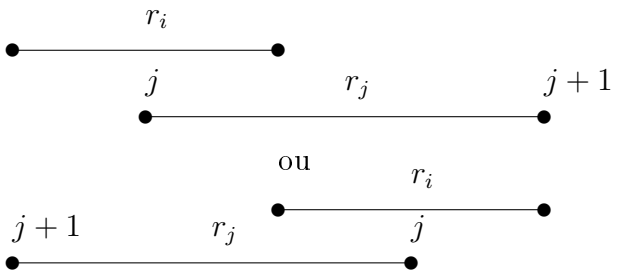
Définition 4.35 *Une inversion élémentaire* r_i chevauche *une inversion élémentaire* r_j *si* r_i *contient soit* j*, soit* $j+1$ *(mais pas les deux).*

Dans notre exemple, r_1 ne contient que 2 et chevauche donc r_2, mais r_0 contient 1 et 2, et ne chevauche donc pas r_1. La proposition suivante montre que la relation de chevauchement est symétrique.

Proposition 4.1 r_i *chevauche* r_j *si et seulement si* r_j *chevauche* r_i*.*

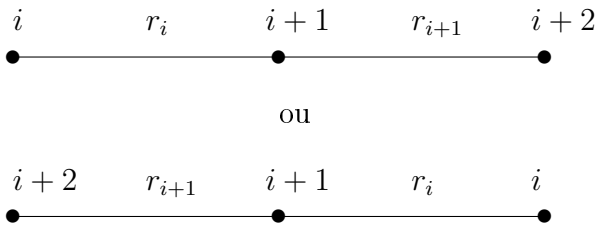
Démonstration : supposons que r_i contient $j \in r_j$ (la démonstration est similaire si r_i contient $j+1 \in r_j$) ; il y a trois cas possibles :

1. $i \neq j \neq i+1$: dans ce cas, l'intervalle couvert par r_j contiendra forcément une borne de r_i ;



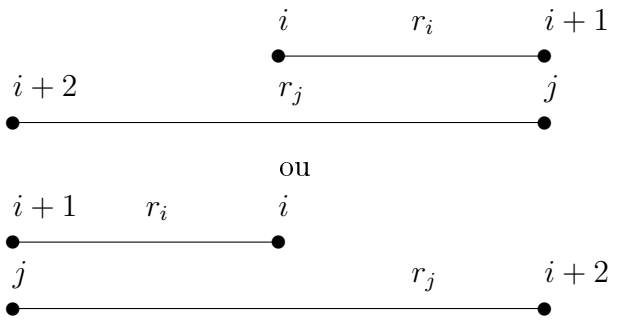
2. $j = i+1$: dans ce cas, comme l'hypothèse a été faite que r_i contenait j, alors r_i contient $i+1$ et il faut montrer que r_{i+1} chevauche r_i ; considérons les trois éléments i, $i+1$ et $i+2$, et les différentes configurations possibles pour ceux-ci :

   (a) si dans la permutation, $i+1$ est situé entre i et $i+2$, alors r_{i+1} ne contient bien qu'une seule borne de r_i : $i+1$



   (b) si dans la permutation, i et $i+2$ se trouvent du même "côté" de $i+1$, alors :
   - soit les éléments considérés apparaissent dans l'ordre $i+2$, i, $i+1$, et dans ce cas $i+1$ est négatif, et comme il suit $i+2$, il sera écarté de r_{i+1} qui ne contiendra alors que la borne i (cf. définition 4.34) ;
   - soit les éléments considérés apparaissent dans l'ordre $i+1$, i, $i+2$, et dans ce cas $i+1$ est positif, et comme il précède $i+2$, il sera écarté de r_{i+1} qui ne contiendra alors que la borne i (cf. définition 4.34) ;

On peut donc en conclure que r_i contient $i+1$ si et seulement si r_{i+1} ne contient pas $i+1$; r_{i+1} chevauche donc bien r_i.



3. $\underline{i = j+1}$: se démontre de la même façon que le cas précédent. □

Définition 4.36 *Une* composante connexe *d'une permutation est une composante connexe du graphe de chevauchement des arêtes.*

La relation de chevauchement de la définition 4.35 est similaire à la relation de chevauchement des arêtes, si ce n'est qu'elle est définie sur les intervalles originaux de la permutation et pas sur la permutation non signée correspondante de $2n$ points. Les graphes de ces deux relations sont isomorphes : en effet, un sommet du graphe de chevauchement correspond à une inversion, et deux inversions sont reliées si et seulement si l'application de l'une modifie l'orientation de l'autre.

Les avantages de la définition de la relation de chevauchement sur les intervalles de la permutation originale résideront dans la possibilité de détecter et d'analyser les composantes connexes du graphe directement par l'inspection de la permutation.

Définition 4.37 *La* portée *d'une composante connexe C est le plus petit intervalle contenant toutes les inversions élémentaires de C.*

Définition 4.38 *Un intervalle commun* encadré *F dans une permutation signée π est un sous-vecteur de π de la forme $(m\ \pi_k\ \cdots\ \pi_j\ M)$ ou $(-M\ \pi_k\ \cdots\ \pi_j\ -m)$ tel que :*

1. *m et M sont respectivement les éléments de valeurs minimale et maximale dans F ;*
2. *$[k-1 \overset{\pi}{—} j+1]$ est un intervalle commun ;*
3. *F n'est pas l'union d'intervalles plus courts vérifiant les deux propriétés précédentes.*

*L'intervalle $[k \overset{\pi}{—} j]$ est appelé l'*intérieur *de l'intervalle commun encadré.*

Théorème 4.5 *Un intervalle est un intervalle commun encadré si et seulement si son intérieur est la portée d'une composante connexe.*

Démonstration : la démonstration ne s'occupera que du cas où l'intervalle est de la forme $(m\ \pi_k\ \cdots\ \pi_j\ M)$, le cas $(-M\ \pi_k\ \cdots\ \pi_j\ -m)$ se démontrant de façon similaire.

1. $\Leftarrow$: aucune inversion élémentaire incluse dans un intervalle commun F ne peut contenir les bornes de cet intervalle : en effet, r_m et r_{M-1} sont incluses dans F et ne contiennent ni m ni M (écartés conformément aux conditions de la définition 4.34), et l'intérieur de F étant un intervalle commun, pour tout élément $m < i < M-1$, $i+1$ appartient également à l'intérieur de F. On peut donc en déduire que toute composante connexe est contenue entièrement dans un intervalle commun encadré.

   Supposons que $I = [k \overset{\pi}{—} j]$ est la portée d'une composante connexe C ; si cette portée est vide, le résultat est trivial. Sinon, par la définition 4.37, il existe au moins une inversion r dans C contenant π_k et une inversion r' contenant π_j. De plus, aucun inversion élémentaire ne peut contenir à la fois un élément de I et un élément externe à I, sinon elle chevaucherait r ou r'.

   Si i est un élément de I, alors r_i et r_{i+1} sont également incluses dans I ; sinon, r_i - par exemple - et I seraient disjoints, et alors $i = \pi_k$ ou $i = \pi_j$, ce qui impliquerait que soit r, soit r' chevauche r_i.

   Donc, $i-1$ et $i+1$ appartiennent à $I' = [k-1 \overset{\pi}{—} j+1]$, ce qui a pour conséquence que π_{k-1} et π_{j+1} sont les éléments extrêmes de l'intervalle, et que $[k-1 \overset{\pi}{—} j+1]$ est un intervalle commun. π_{k-1} et π_{j+1} n'appartenant pas à la portée de C, ils sont respectivement égaux à m et M (ou $-M$ et $-m$). Finalement, I' ne peut pas être l'union d'intervalles encadrés communs plus courts puisqu'une composante connexe doit se trouver dans un seul intervalle commun encadré.
2. $\Rightarrow$: de l'autre côté, si $F = (m \cdots M)$ est un intervalle encadré commun, alors la portée de la composante de r_m est encadrée par $(m \cdots M')$, avec $M' \leq M$; mais si $M' < M$, alors l'intervalle $(M' \cdots M)$ est un intervalle commun commençant par sa valeur minimale et terminant par sa valeur maximale, et $F = (m \cdots M') \cup (M' \cdots M)$. □

L'inclusion des portées induit un ordre partiel sur les composantes connexes de portées non vides. Une composante connexe contient une inversion élémentaire non vide r si r est incluse dans la portée de C, mais pas dans celles des composantes inférieures à C.

Définition 4.39 *Une composante connexe de portée non vide est* orientée *si elle contient au moins une inversion élémentaire orientée,* non orientée *sinon.*

Définition 4.40 *Les* éléments *d'une composante connexe constituent l'ensemble des bornes des inversions élémentaires qu'elle contient.*

Proposition 4.2 *Une composante connexe est non orientée si et seulement si tous ses éléments ont le même signe dans π.*

Démonstration : il est évident que si une composante est orientée, ses éléments n'ont pas tous le même signe. Soit une composante C non orientée avec les extrema m et M (non signés) ; ces deux valeurs sont des éléments de la composante connexe puisque r_m et r_{M-1} appartiennent à la composante. Les éléments de C sont tous des entiers allant de m à M, avec des "trous" correspondant aux portées des composantes

plus petites que C. Comme les deux éléments encadrant ces composantes sont de même signe, tout changement de signe dans la séquence ordonnée des éléments de C doit se produire dans cette séquence. □

Nous allons maintenant présenter les deux algorithmes qui serviront à calculer la distance d'une permutation ; tout comme dans [3], nous allons d'abord décrire le cas des permutations positives.

L'algorithme 4.1 identifie les intervalles communs encadrés dans une permutation positive et se base sur le lemme suivant.

Lemme 4.16 *Si $F = (m\ \pi_k\ \cdots\ \pi_j\ M)$ est un intervalle commun encadré d'une permutation positive, alors chacun des π_i (avec $k \leq i \leq j$) possède un élément dans l'intérieur de F qui soit lui est inférieur et le suit, soit lui est supérieur et précède.*

Démonstration : procédons par l'absurde : si tous les éléments inférieurs à π_i le précèdent et tous les éléments supérieurs à π_i le suivent, alors $(m\ \cdots\ \pi_i)$ et $(\pi_i\ \cdots\ M)$ sont des intervalles encadrés, et F en est l'union, ce qui contredit sa définition. □

Définissons M_i comme l'élément de la permutation qui est le plus proche de π_i, qui précède π_i et qui lui est supérieur ($M_i = n$ si un tel élément n'existe pas).

Dans l'algorithme 4.1, S est un *stack* dont le sommet est noté s.

Algorithme 4.1 Détection des intervalles communs encadrés

Données: une permutation π
Résultat: une liste contenant tous les intervalles communs encadrés de π

```
1:  S.Push(0) ;
2:  for i = 1 à n do
3:     while π_i < π_s ou π_i > M_s do
4:        S.Pop() ;
5:     end while
6:     if [s —π— i] est un intervalle commun encadré then
7:        l'ajouter à la liste des intervalles communs encadrés ;
8:     end if
9:     S.Push(i) ;
10: end for
```

La proposition suivante certifie que tout intervalle commun encadré de la permutation π sera testé.

Proposition 4.3 *Si $[s \overset{\pi}{—} j]$ est un intervalle commun encadré, l'indice s ne sera pas dépilé avant que j ne soit empilé.*

Démonstration : π_s étant par définition le plus petit élément de l'intervalle $[s \overset{\pi}{—} j]$, tout élément de $[s+1 \overset{\pi}{—} j]$ est supérieur à π_s, donc aucun de ces éléments ne peut vérifier la première condition du `while` de l'algorithme 4.1. De plus, tout élément entre π_s et π_j est inférieur à M_s, puisque M_s doit être au moins supérieur à π_j :

en effet, si $\pi_s < M_s < \pi_j$, alors $[s \overset{\pi}{—} j]$ ne serait pas un intervalle commun encadré puisqu'il manquerait l'élément M_s, situé à gauche de π_s. Donc, aucun élément entre π_s et π_j ne peut dépiler s. □

Proposition 4.4 *Tous les indices i tels que $s < i < j$ seront dépilés avant que j ne soit empilé.*

Démonstration : soit $s < i < j$; par le lemme 4.16, nous savons que π_i possède un élément :

- soit inférieur le suivant, et dans ce cas cet élément dépilera i (conformément à la première condition du `while` de l'algorithme 4.1) ;
- soit supérieur le précédant dans l'intervalle $[s \overset{\pi}{—} i]$, et dans ce cas, comme $M_i < \pi_j$, si i reste sur le *stack* jusqu'à la fin, π_j le dépilera (conformément à la seconde condition du `while` de l'algorithme 4.1). □

Il se pose maintenant la question de savoir comment déterminer si $[s \overset{\pi}{—} i]$ est un intervalle commun encadré (ligne 6 de l'algorithme de l'algorithme 4.1) ; le premier test élémentaire à effectuer est de compter le nombre d'éléments entre ces deux indices, puisqu'une condition nécessaire pour que $[s \overset{\pi}{—} i]$ soit un intervalle commun encadré est :

$$\pi_i - \pi_s = i - s$$

Si s est le sommet de S, alors tous les éléments dans l'intervalle $[s+1 \overset{\pi}{—} i]$ sont supérieurs à π_s. Il faut également vérifier que seules des valeurs inférieures à π_i se trouvent dans $[s+1 \overset{\pi}{—} i]$, ce qui se fait en mémorisant l'élément maximal rencontré entre deux indices empilés consécutifs.

L'algorithme 4.2 calcule les M_i dont nous avions besoin dans l'algorithme 4.1 ; ici, S est toujours un *stack* mais qui contient au départ la valeur n.

La correction de cet algorithme est basée sur le fait que, si $M_i = \pi_k$, alors M_i est - par définition - supérieur à toutes les valeurs de l'intervalle $[k+1 \overset{\pi}{—} i]$ et sera donc empilé lorsque π_{k+1} sera lu et qu'aucune valeur de l'intervalle ne peut dépiler M_i. Mais π_i dépilera toutes les valeurs dans l'intervalle dès que l'occasion se présentera (le même argument est utilisé pour $M_i = n$, en prenant $k = 0$). Ces résultats sont résumés dans le théorème suivant :

Théorème 4.6 *Tous les intervalles communs encadrés d'une permutation positive peuvent être trouvés par les deux algorithmes précédents en temps et en espace $O(n)$*

Examinons maintenant le cas des permutations signées ; l'adaptation de l'algorithme 4.1 n'est pas compliquée, puisque tout intervalle commun encadré de bornes positives d'une permutation signée π sera l'union d'intervalles communs encadrés de sa version positive π^+. Nous allons en fait utiliser l'algorithme 4.1 sur π^+, mais en évitant d'empiler des éléments négatifs. La correction de cette méthode est justifiée par le lemme suivant, qui est une extension du lemme 4.16 et se démontre de façon similaire.

Algorithme 4.2 Calcul des M_i

Données: une permutation π

Résultat: un vecteur contenant, pour chaque élément i, la valeur M_i associée

```
1:  S.Push(n) ;
2:  M_0 ← n ;
3:  for i = 1 à n do
4:     if π_{i-1} > π_i then
5:        M_i ← π_{i-1} ;
6:        S.Push(π_{i-1}) ;
7:     else
8:        while s < π_i do
9:           S.Pop() ;
10:       end while
11:       M_i ← s ;
12:    end if
13: end for
```

Lemme 4.17 *Si $F = (m\ \pi_k\ \cdots\ \pi_j\ M)$ est un intervalle commun encadré d'une permutation positive, alors chacun des π_i (avec $k \leq i \leq j$) possède un élément dans l'intérieur de F qui soit lui est inférieur et le suit, soit lui est supérieur et précède ; sinon, π_i est négatif.*

Ce lemme peut être utilisé pour démontrer les équivalents des propositions 4.3 et 4.4, en se rappelant bien que les éléments négatifs ne sont pas empilés mais que leurs valeurs absolues peuvent être utilisées pour dépiler des éléments.

L'identification des composantes de la forme $(-M\ \cdots\ -m)$ peut se faire en "inversant" l'algorithme[4], ce qui permet de détecter les composantes connexes de la permutation en une seule passe, à l'aide de quatre *stacks* (cf. code source).

Détecter si tous les éléments d'une composante sont de même signe peut se faire par un marquage approprié sur le sommet du *stack* ; dans [3], une version pour les composantes de portée $(m\ \cdots\ M)$ a été présentée : dans ce cas, la composante sera non orientée si tous les éléments négatifs de l'intervalle sont "protégés" comme il se doit par les cadres positifs de composantes inférieures. Le marquage se fait comme suit :

1. si π_i est négatif, marquer le sommet du *stack* ;
2. si un élément marqué est retiré du *stack*, marquer le nouveau sommet du *stack* ;
3. si $[s \overset{\pi}{—} i]$ est une composante connexe, démarquer s.

Comme nous l'avons vu à la fin de la section 4.3, le nombre minimal d'inversions nécessaires au tri de la permutation $\pi = (0\ \pi_1\ \cdots\ \pi_{n-1}\ n)$ est

$$d(\pi) = n - c + h + f$$

[4]Bergeron [3] dit que l'on peut aussi inverser π mais cela n'est pas un terme approprié, car elle ne fait pas référence à π^{-1} mais simplement à la permutation obtenue en lisant les éléments de droite à gauche (sauf les bornes) et en inversant leur signe - ou dit plus simplement : en inversant l'intervalle $[1 \overset{\pi}{—} n-1]$.

Les notions sur lesquelles se base cette formule ont été introduites en section 4.3, mais elle va à présent se baser sur les nouvelles notions liées aux intervalles communs. Soit l'ensemble des composantes non orientées comportant plus de deux éléments dans la permutation ; ces composantes sont ordonnées partiellement par l'inclusion des portées. Chaque élément minimal de cet ordre est une *haie*. De plus, l'élément maximal est une haie si elle existe et qu'aucun de ses éléments ne se trouve entre deux haies (ceci est conforme à la définition 4.14). Les définitions suivantes permettent de faire le lien avec ce qui a été présenté dans la section 4.3.

Définition 4.41 *Deux inversions élémentaires sont* liées *si elles sont consécutives, ou si l'une est le préfixe ou le suffixe de l'autre.*

Définition 4.42 *Un* cycle *est une chaîne fermée d'inversions liées.*

Reprenons l'exemple de la définition 4.34 :

$$(0\ -6\ 3\ -4\ 5\ 2\ -1\ \boxed{7\ 9\ 8\ 10}\ )$$

Nous avons :
- $h = 1$; l'unique haie est encadrée et n'est pas une super haie, puisque son tri donne la permutation $(0\ {-6}\ 3\ {-4}\ 5\ 2\ {-1}\ 7\ 8\ 9\ 10)$ qui ne comporte plus aucune haie. Nous pouvons donc en déduire que π n'est pas une forteresse et donc $f = 0$.
- $c = 4$; les inversions élémentaires liées étant :
  1. r_0 et r_6 ;
  2. r_3 et r_4 ;
  3. r_1 et r_2 ;
  4. r_5 et r_2 ;
  5. r_7 et r_8 ;
  6. r_8 et r_9 ;

  qui forment les cycles $\{r_0, r_6\}$, $\{r_3, r_4\}$, $\{r_1, r_2, r_5\}$ et $\{r_7, r_8, r_9\}$.

Et nous concluons de ces observations que $d(\pi) = 7$.

Le problème principal dans le calcul de $d(\pi)$ est de savoir si une composante non orientée est une haie, et si c'est le cas, si c'est de plus une super haie. L'identification des haies qui sont des éléments minimaux est facile (l'algorithme donné dans l'annexe énumère les composantes de gauche à droite, pour les composantes dont la portée n'est pas incluse dans une autre, et des composantes internes vers les composantes externes pour celles dont les portées sont emboîtées).

Une composante non orientée U est une haie s'il s'agit de la première identifiée par l'algorithme, ou si sa portée ne contient pas la portée d'une composante non orientée précédente. Si la permutation ne possède qu'une haie minimale, la composante non orientée maximale, si elle existe, sera aussi une haie.

Pour les haies qui sont des éléments minimaux, une manière simple de voir si elles sont des super haies est de tester si leur ancêtre immédiat dans l'ordre partiel ne contient qu'une composante orientée. Cela n'est utile que quand la permutation

possède au moins trois haies. Une haie U est une super haie si la portée de la composante non orientée suivante contient la portée de U et ne contient pas celle de la haie précédant U. Pour tester si la composante maximale est une haie ou une super haie, nous appliquons l'algorithme de l'annexe aux éléments de la permutation qui ont été décalés, en utilisant comme nouvel élément 0 un élément de la dernière haie minimale.

4.6 Applications

De façon plus générale, le problème du tri d'une permutation en un nombre minimal d'opérations est connu sous le nom de *réarrangement de génôme* et consiste à trouver une série de réarrangements transformant un génôme en un autre, de façon à déterminer des scénarios possibles d'évolution entre différentes espèces ; l'étude de l'ordre des gènes est justifiée biologiquement par le fait que plusieurs espèces peuvent très bien avoir un grand nombre de gènes en commun, et ne différer principalement que par l'ordre de ces gènes.

La comparaison entre deux espèces peut s'effectuer à plusieurs niveaux, et la technique traditionnelle consiste à comparer les gènes, de façon à reconstruire des *arbres phylogéniques*, qui sont des arbres binaires représentant graphiquement des scénarios possibles d'évolution entre diverses espèces. Signalons que les arbres phylogéniques *peuvent* comporter une racine, représentant l'ancêtre commun de tous les éléments, mais qu'il existe également des arbres sans racine : ce second cas est plus facile à traiter, car il est en général assez difficile de déterminer l'ancêtre commun de tous les éléments d'un ensemble (la plupart des programmes construisant ces arbres ne produisent d'ailleurs que des arbres sans racine).

Expliquons comment une distance, par exemple la distance des inversions, peut être utilisée pour construire un arbre possédant une racine : soit un ensemble de permutations signées, représentant des génômes. Il faut calculer la distance[5] entre toute paire d'éléments distincts de cet ensemble, puis construire une matrice de distances M dont la composante $M_{\pi\sigma}$ contient $d(\pi, \sigma)$: cette matrice est symétrique, et les éléments de sa diagonale principale sont nuls. Divers arbres peuvent ensuite être déduits de cette matrice, selon différentes méthodes, en groupant les permutations sur base de leurs distances.

La figure 4.14, extraite de [17], montre un exemple d'un tel arbre ; les feuilles représentent les espèces étudiées, et deux éléments dérivés d'un ancêtre commun partagent le même nœud. La longueur des branches est déterminée par la distance entre les éléments reliés.

En réalité, l'utilisation d'une distance n'est qu'un exemple parmi d'autres de méthode de constructions des arbres phylogéniques, avec chacune des avantages, des inconvénients et des domaines d'applications différents. Nous renvoyons le lecteur à [8, 17] pour plus d'informations.

[5] La distance des inversions n'est qu'un exemple parmi d'autres distances pouvant être utilisées.

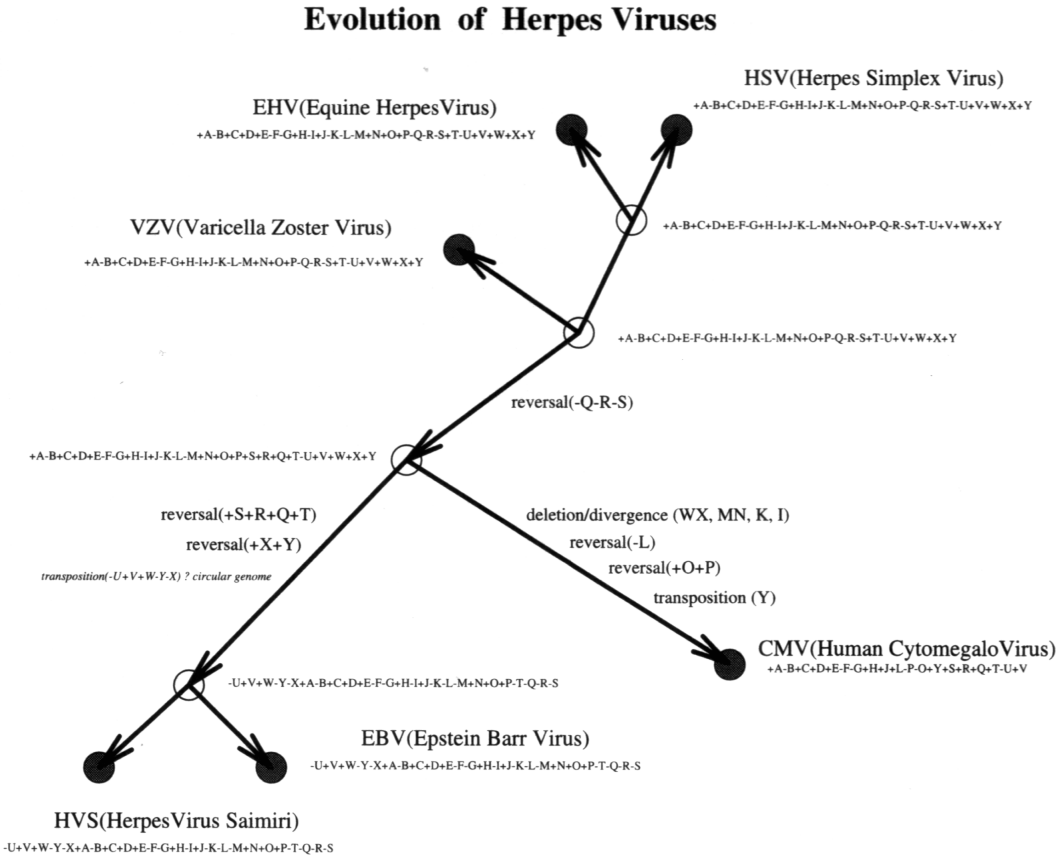


FIG. 4.14 – Un arbre phylogénique (source : [17])

Chapitre 5

Etude statistique de la distance des inversions entre permutations signées

Ce chapitre présente quelques résultats personnels concernant la distance des inversions sur les permutations signées. Des cas particuliers ont été obtenus expérimentalement par l'implémentation de l'algorithme présenté dans [3] ; ensuite, j'ai construit des démonstrations des énoncés généraux. Voici les principales contributions énumérées dans ce chapitre :

- des résultats de dénombrement concernant le nombre de permutations situées à une certaine distance de l'identité, et plus précisément les permutations positives à distance 0, 1, 2 et 3 (proposition 5.2), ainsi que sur les permutations non nécessairement positives à distance 1 (proposition 5.3), 2 (proposition 5.6) et enfin, une borne inférieure sur le nombre de permutations négatives à distance k (proposition 5.5) ;
- la mise en évidence de l'existence de deux permutations de forme particulière : la permutation χ (proposition 5.1), qui est une des permutations permettant d'atteindre la valeur maximale de la distance des inversions et les permutations pseudo-identitaires (section 5.3), qui permettent de fournir une minoration sur le nombre de permutations signées à une certaine distance. Le calcul de la distance de ces deux permutations est également donné (voir la proposition 5.4 pour les permutations pseudo-identitaires) ;
- l'introduction de la notion de symétrique d'une inversion par rapport à un intervalle (définition 5.2) ;
- l'utilisation de cette notion pour montrer l'existence de plusieurs séquences d'inversions permettant d'arriver à la même permutation (lemmes 5.1, 5.2 et 5.3, ainsi que le corollaire 5.1) ;
- une observation sur la concaténation d'éléments "en bonne place" à une permutation (lemme 5.4).

Signalons enfin que les résultats obtenus expérimentalement, qui ont servi de bases aux différentes propositions et lemmes établis dans ce chapitre, sont également inclus dans ce dernier sous forme de tables.

5.1 Explications sur le code source

Je vais expliquer dans cette section les grandes lignes de l'implémentation de mon programme, qui contient entre autres l'algorithme de Bergeron [3]. Signalons que la permutation passée à la classe se chargeant de calculer sa distance sera transformée en une permutation positive et que ses signes seront stockés à part, ce qui permet de simplifier un peu les choses.

5.1.1 Détection des cycles

Un cycle, comme nous l'avons vu au chapitre précédent, est défini ici comme une chaîne fermée d'inversions liées ; informellement, il suffit donc de parcourir ces inversions en respectant cette relation jusqu'au moment où l'on revient au point de départ. Voici une manière plus simple de voir les choses[1] pour l'implémentation : considérons que chaque élément possède un point à sa gauche et un point à sa droite (sauf bien entendu le premier, qui ne possède un point qu'à sa droite, et le dernier, qui ne possède un point qu'à sa gauche). La définition 4.34 est alors reformulée de la façon suivante :

Définition 5.1 *Soit une permutation signée π, encadrée par 0 et n ; à chaque élément $0 \leq |i| \leq n-1$ est associée une* inversion élémentaire $r_{|i|}$, *agissant sur l'intervalle compris entre :*

- *le point à droite de l'élément i si i est positif, le point à gauche de i dans le cas contraire ;*
- *le point à gauche de l'élément $i+1$ si $i+1$ est positif, le point à droite de $i+1$ dans le cas contraire.*

Un vecteur de booléens `points` représentera les points à visiter (vrai) ou déjà visités (faux) : `points[i]` représente donc le point situé à droite de l'élément i. Pour chaque point non encore visité, il faut parcourir le cycle auquel il appartient : il faut donc sauver sa position et déterminer les inversions à parcourir.

L'idée de l'algorithme est donc la suivante :

- pour chaque élément i à visiter, examiner son signe et l'origine (c'est-à-dire déterminer si l'on part du point à sa gauche ou à sa droite) : ces deux paramètres détermineront si l'inversion élémentaire qui est en train d'être parcourue est r_i ou r_{i-1} ;
- une fois l'inversion élémentaire déterminée, il faut trouver l'extrémité correspondant au cas examiné, c'est-à-dire qu'il faut trouver la position de l'élément $i+1$ si l'inversion élémentaire examinée est r_i et la position de l'élément $i-1$ si l'inversion élémentaire examinée est r_{i-1} ;
- enfin, le signe de l'extrémité trouvée détermine l'élément suivant à examiner, ainsi que l'origine. Il faut bien entendu marquer son passage, donc annuler la composante correspondante dans le vecteur `points`.

[1] Ce qui suit provient d'un échange d'e-mails avec Anne Bergeron et n'est pas décrit dans l'article où est fourni l'algorithme implémenté.

Ces étapes sont à répéter tant que l'on ne retombe pas sur la position i de départ ; quand cela arrive, tout le cycle a été parcouru : le nombre de cycles est donc incrémenté de 1 et l'algorithme se poursuit avec le prochain point non encore visité (s'arrêtant donc quand un tel point n'existe plus). Cette détection est bel et bien linéaire, puisqu'une permutation de n éléments comporte n inversions élémentaires, et qu'étant donné une extrémité i, l'autre extrémité $i \pm 1$ peut être trouvée en temps constant à l'aide de la permutation de π construite avant la détection des cycles (cette autre extrémité se trouve donc à la position $|\pi^{-1}_{|\pi_i|+1}|$). Voici un exemple d'application de cet algorithme :

ι	0	1	2	3	4	5	6	7	8	9	10
π	0 ·	−6 ·	3 ·	−4 ·	5 ·	2 ·	−1 ·	7 ·	8 ·	9 ·	10
π^{-1}	0	−6	5	−2	3	4	−1	7	8	9	10

Le départ se fait en 0, à partir de son point à droite que nous marquons : comme nous partons d'un élément positif et du point à sa droite, nous cherchons r_0 et devons donc trouver 1, situé en $|\pi_1^{-1}| = 6$. Comme -1 est négatif, l'inversion r_0 se poursuit jusqu'au point à sa droite, que nous marquons avant de nous rendre à l'élément à droite de -1. Cet élément est 7, il est donc positif, et comme nous partons du point à sa gauche (celui que nous venons de marquer), nous cherchons r_6 : il faut donc trouver 6, qui est en position $|\pi_6^{-1}| = 1$. Il s'agit en réalité de -6, et comme cet élément est négatif, r_6 se prolonge jusqu'à sa gauche : il faut donc marquer ce point, et comme nous retombons sur notre point de départ nous avons trouvé un cycle. Après avoir trouvé ce premier cycle, l'état de la permutation π est donc le suivant : $(0\surd - 6 \cdot 3 \cdot -4 \cdot 5 \cdot 2 \cdot -1\surd\ 7 \cdot 8 \cdot 9 \cdot 10)$, où $\cdot$ est un point non marqué et $\surd$ est un point marqué.

Examinons ensuite le point suivant non visité : il est situé à droite de -6, qui est un élément négatif, donc nous cherchons r_5. Après avoir marqué le point à droite de -6, nous trouvons par la permutation inverse que 5 est situé en $\pi_{\pi_5^{-1}}$, c'est-à-dire en quatrième position. Comme il s'agit d'un élément positif, nous marquons le point à sa droite et passons à l'élément à sa droite, à savoir 2. Comme 2 est positif et que l'on vient du point à sa gauche, nous cherchons r_1 dont nous trouvons la position de la borne inférieure dans π grâce à $|\pi_1^{-1}| = 6$. -1 étant négatif, il faut marquer le point à sa gauche et revenir sur l'élément 2. Comme le départ se fait à nouveau de 2, qui est positif, mais cette fois-ci à partir du point à sa droite, nous cherchons r_3 ; nous trouvons aisément $|\pi_3^{-1}| = 2$, et 3 étant positif, l'intervalle se prolonge jusqu'au point à sa gauche, qui est notre point de départ ; un deuxième cycle a ainsi été identifié, et le même procédé doit être répété sur la permutation $(0\surd - 6\surd\ 3 \cdot -4 \cdot 5\surd\ 2\surd - 1\surd\ 7 \cdot 8 \cdot 9 \cdot 10)$. On procède de la même manière pour les deux cycles restants (respectivement $\{r_4, r_5\}$ et $\{r_7, r_8, r_9\}$).

En réalité, dans mon implémentation, je calcule la permutation inverse de la version non signée de π, ce qui nous évite de devoir manipuler des valeurs absolues inutilement puisque π^{-1} ne nous sert qu'à trouver la position d'un élément.

5.1.2 Compte du nombre de haies

Enumération des composantes non orientées

Cela se fait grâce à l'algorithme donné dans l'annexe de [3], qui a été expliqué au chapitre précédent.

Identification des haies

Une composante non orientée est une haie s'il s'agit de la première composante non orientée identifiée par l'algorithme ou si sa portée ne contient pas celle d'une composante non orientée identifiée précédemment. Le cas de la composante maximale est plus compliqué, car il faut tester s'il n'existe pas d'éléments entre deux haies minimales précédemment identifiées.

Tester si la composante maximale est une haie revient à tester si des éléments se trouvent entre deux haies ; pour ce faire, il faut maintenir deux variables L(eft) et R(ight) contenant les bornes de la première haie identifiée ; à chaque fois que l'on identifie une nouvelle haie, il faudra comparer ses bornes à L et R : soit la nouvelle haie chevauche l'ensemble délimité par $[L, R]$ et dans ce cas il suffit de mettre à jour ces variables, soit elle suit cet ensemble en ne laissant aucun "trou" (il suffit donc de mettre à jour R), soit elle le suit en laissant un trou et il faut mettre la variable booléenne `holes` à `true`. Lorsque vient le moment d'examiner la composante maximale (si elle existe), ce sera une haie si et seulement si `holes = false`.

Identification des super haies

Ce test s'effectuera à chaque fois que l'on aura identifié une haie : une haie est une super haie si la portée de la composante non orientée suivante contient la portée de cette haie et ne contient pas la portée de la composante non orientée précédant cette haie. On stockera dans une variable le nombre de super haies de la permutation.

Test de la propriété de forteresse

La permutation est une forteresse si le nombre de haies qu'elle contient est impair et égal au nombre de super haies identifiées.

5.1.3 Génération des permutations

La génération des permutations encadrées positives se fait par l'algorithme récursif de Heap [11], qui est le plus rapide à ma connaissance [18]. Ensuite, pour chaque permutation positive ainsi générée, stockée dans le vecteur `values`, une autre fonction (`sign_permutation()`) se chargera de générer les 2^{n-2} manières de placer les signes $-$ ainsi que de calculer sa distance et de stocker les caractéristiques qui nous intéressent. Cette dernière fonction procède comme suit : à chaque permutation est attribué un numéro (entre 0 et $2^{n-2} - 1$), et la permutation suivante est générée à partir de la permutation courante, en changeant les signes des éléments dont les positions correspondent aux positions des bits qui changent lors de l'incrémentation du compteur c servant à numéroter les permutations. Ces positions sont aisément

n	Moyenne	Variance	Valeur maximale
3	0	0	0
4	1.5	0	3
5	2.5	6,25	3
6	3.5416667	28,9149303	5
7	4.4	285,09	6
8	5.33194445	1686,9606461	7
9	6.19107148	15401,9057529	8
10	7.11252480	128810,957650	9

TAB. 5.1 – Propriétés des distances des permutations positives encadrées de n éléments, pour n entre 3 et 10

obtenues en assignant à une variable `mask` la valeur $c \oplus c+1$ (où $\oplus$ représente le OU exclusif bit à bit). Il suffit ensuite de multiplier par -1 les positions de `values` correspondant aux bits à 1 de `mask`.

5.2 Observations sur les permutations encadrées positives

Le tableau 5.1 reprend les résultats obtenus pour les permutations positives encadrées, avec n variant de 3 à 10 ; on peut remarquer que $d(\pi)$ est majorée par $n-1$ (et bien entendu minorée par 0, cf. l'identité), sauf si n vaut 3 ou 5, auquel cas elle est majorée par $n-2$ (démontré dans [16]).

Je propose les notations :

1. $\left[\!\left[{n \atop k} \right]\!\right]^+$ pour le nombre de permutations positives encadrées de n éléments telles que $d(\pi) = k$;
2. $\left[\!\left[{n \atop k} \right]\!\right]^-$ pour le nombre de permutations comportant au moins un élément négatif encadrées de n éléments telles que $d(\pi) = k$;
3. $\left[\!\left[{n \atop k} \right]\!\right]$ pour le nombre de permutations encadrées, positives ou non, de n éléments telles que $d(\pi) = k$, et naturellement $\left[\!\left[{n \atop k} \right]\!\right] = \left[\!\left[{n \atop k} \right]\!\right]^+ + \left[\!\left[{n \atop k} \right]\!\right]^-$.

En examinant le tableau 5.2, on peut faire les observations suivantes sur $\left[\!\left[{n \atop k} \right]\!\right]^+$:

1. $\forall n > 0 : \left[\!\left[{n \atop 0} \right]\!\right] = 1$; en effet, une seule permutation se trouve à distance 0 de la permutation identité et il s'agit de cette dernière ;
2. $\forall n > 0 : \left[\!\left[{n \atop 1} \right]\!\right]^+ = 0$; en effet, comme les inversions changent le signe des éléments, il est impossible de partir d'une permutation positive et d'arriver à l'identité avec une seule inversion ;
3. $\forall n > 0 : \left[\!\left[{n \atop 2} \right]\!\right]^+ = 0$; s'il était possible, à l'aide de deux inversions sur une permutation positive, de retrouver l'identité, cela signifierait que l'une annule l'effet de l'autre, donc que la permutation de départ est l'identité, de distance nulle ;

4. $\forall n \geq 4 : \left[\!\left[{n \atop 3} \right]\!\right]^+ = \binom{n}{n-4} = \binom{n}{4}$; on peut constater cela en comparant le triangle de Pascal (tableau 5.3) et le tableau 5.2, disposé de la même façon et reprenant les fréquences des permutations positives encadrées de n éléments à distance k (voir proposition 5.2 pour la démonstration).

Avant de présenter mes démonstrations, voici deux observations supplémentaires que je n'ai pas réussi à démontrer :

1. la distance de plus grande fréquence est $n-2$ si n est impair, $n-3$ sinon ;
2. $\forall n \geq 0 : \left[\!\left[{n \atop 4} \right]\!\right]^+ = 0$.

Je vais maintenant démontrer qu'une permutation de forme particulière possède une distance caractéristique ; pour cela, je vais utiliser la formule évoquée au chapitre précédent, donnant la distance d'une permutation de $n-2$ éléments :

$$d(\pi) = n - 1 - c(\pi) + h(\pi) + f$$

Proposition 5.1 *Soit* $\chi = (0\ n-2\ n-3\ \cdots\ 2\ 1\ n-1)$ *; alors :*

$$d(\chi) = n - 1 - (n\ mod\ 2)$$

Démonstration :

- $c = 1 + (n \bmod 2)$: si n est pair, les inversions élémentaires r_1, r_2, ..., r_{n-3} sont de longueur 2 et comprennent leurs bornes (en effet, celles-ci sont positives et $i+1$ précède toujours i pour $1 \leq i \leq n-3$) qui sont contiguës ; parmi ces inversions, celles d'indice impair forment une chaîne d'inversions liées (r_1 est consécutive à r_3, qui est consécutive à r_5, ... et ainsi de suite jusqu'à r_{n-3}). Il en est de même pour celles d'indice pair, qui forment une seconde chaîne d'inversions liées, mais ces deux chaînes ne sont pas fermées ; or, r_{n-2} a pour préfixe r_{n-4} et pour suffixe r_1, ce qui permet de joindre les deux chaînes. Et comme r_0 a pour préfixe r_{n-3} et pour suffixe r_2, elle joint les deux extrémités de cette chaîne et crée l'unique cycle de χ.
  Dans le cas où n est impair, la démonstration est similaire, si ce n'est que les inversions préfixe et suffixe de r_{n-2} sont toutes deux d'indice impair et que les inversions préfixe et suffixe de r_0 sont toutes deux d'indice pair, ce qui crée bien deux cycles distincts.
- $h = 1$: par définition de χ, tous les éléments de 1 à $n-2$ sont consécutifs et dans l'ordre décroissant, donc il ne peut y avoir aucune haie sur l'intervalle $[1 \xrightarrow{\chi} n-2]$; la seule haie est donc la composante maximale.
- $f = 0$: en effet, la composante maximale est la seule haie et n'est pas une super haie, puisqu'une fois ses éléments triés, on a la permutation identité qui ne comporte aucune haie.

Injectons ces données dans la formule donnant $d(\chi)$; on a :

$$\begin{aligned} d(\chi) &= n - 1 - c + h + f \\ &= n - 1 - 1 - (n \bmod 2) + 1 + 0 \\ &= n - 1 - (n \bmod 2) \end{aligned}$$

□

La figure 5.1 montre deux exemples illustrant les deux cas possibles pour χ.

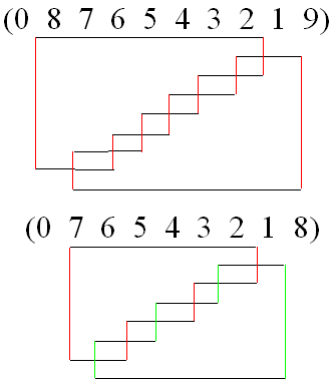


FIG. 5.1 – La première permutation comporte un cycle (en rouge), la seconde en comporte deux (en rouge et en vert)

J'introduis à présent la notion de *symétrique* d'une inversion, qui me sera très utile pour démontrer mes résultats suivants.

Définition 5.2 *Soit une inversion $\rho(i_\rho, j_\rho)$ et un intervalle $I = [i_I, j_I]$, avec $i_\rho \geq i_I$ et $j_\rho \leq j_I$; le* symétrique *de ρ par rapport à I est l'inversion $\hat{\rho}(i_I + j_I - j_\rho, j_I - i_\rho + i_I)$*

Exemple :

$$\overbrace{i_I \quad \overbrace{i_\rho \ \ \rho \ \ j_\rho}^{I} \quad j_I}^{} \Rightarrow \overbrace{i_I \quad \overbrace{i_{\hat{\rho}} \ \ \hat{\rho} \ \ j_{\hat{\rho}}}^{I} \quad j_I}^{}$$

Par abus de langage, on parlera également de symétrique d'une inversion ρ_1 par rapport à une inversion ρ_2 pour désigner le symétrique de ρ_1 par rapport à l'intervalle sur lequel agit ρ_2. Une notation plus rigoureuse consisterait à reprendre l'intervalle par rapport auquel se fait la symétrie, par exemple $\hat{\rho}^I$ ou $\hat{\rho}^{i_I, j_I}$; mais, pour alléger celle-ci, on préférera écrire simplement $\hat{\rho}$, et le contexte permettra de voir clairement quel est l'intervalle I. Remarquons que si ρ couvre l'intervalle I, alors $\hat{\rho} = \rho$. Cette notion de symétrique d'une inversion va nous servir dans le décompte des permutations à distance donnée de l'identité. Elle apparaît déjà dans le lemme suivant.

Lemme 5.1 *Si ρ_1 et ρ_2 sont des inversions liées, alors il existe deux inversions liées ρ_3 et ρ_4 avec $(\rho_1, \rho_2) \neq (\rho_3, \rho_4)$ telles que*

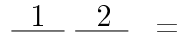
$$\rho_2 \circ \rho_1 = \rho_4 \circ \rho_3$$

Démonstration : deux inversions $\rho_1(i_1, j_1)$ et $\rho_2(i_2, j_2)$, de longueurs respectives $l_1 = j_1 - i_1 + 1$ et $l_2 = j_2 - i_2 + 1$, sont liées si :

n	$\left[\!\left[{n \atop 0} \right]\!\right]^+$	$\left[\!\left[{n \atop 1} \right]\!\right]^+$	$\left[\!\left[{n \atop 2} \right]\!\right]^+$	$\left[\!\left[{n \atop 3} \right]\!\right]^+$	$\left[\!\left[{n \atop 4} \right]\!\right]^+$	$\left[\!\left[{n \atop 5} \right]\!\right]^+$	$\left[\!\left[{n \atop 6} \right]\!\right]^+$	$\left[\!\left[{n \atop 7} \right]\!\right]^+$	$\left[\!\left[{n \atop 8} \right]\!\right]^+$	$\left[\!\left[{n \atop 9} \right]\!\right]^+$	$\left[\!\left[{n \atop 10} \right]\!\right]^+$
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0
4	1	0	0	1	0	0	0	0	0	0	0
5	1	0	0	5	0	0	0	0	0	0	0
6	1	0	0	15	0	8	0	0	0	0	0
7	1	0	0	35	0	81	3	0	0	0	0
8	1	0	0	70	0	445	24	180	0	0	0
9	1	0	0	126	0	1761	108	2980	64	0	0
10	1	0	0	210	0	5624	361	25199	857	8068	0
11	1	0	0	330	0	15402	999	146813	6041	191174	2120
12	1	0	0	495	0	37542	2421	666590	30029	2244730	42064
13	1	0	0	715	0	83490	5313	2522101	118544	17704745	422774
14	1	0	0	1001	0	172392	10791	8290905	396153	106508788	2902206

TAB. 5.2 – Quelques valeurs expérimentales de $\left[\!\left[{n \atop k} \right]\!\right]^+$

1. elles sont consécutives : dans ce cas, il est trivial que le fait d'appliquer d'abord la première ou la seconde revient au même, puisqu'elles n'ont aucun élément en commun et n'influent donc pas l'une sur l'autre. De façon graphique (le chiffre dénotant l'ordre dans lequel sont appliquées les inversions) :



   On a alors $(\rho_2 \circ \rho_1)\pi = (\rho_1 \circ \rho_2)\pi$, donc dans ce cas ρ_1 et ρ_2 commutent.

2. l'une est préfixe de l'autre : supposons que ρ_1 est préfixe de ρ_2. Dans ce cas, on a une situation qui peut se représenter graphiquement de la façon suivante :

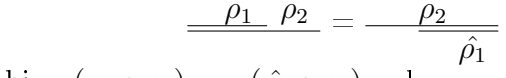
   Dans cette représentation, les inversions s'appliquent toujours "de haut en bas" (comprendre qu'ici on appliquera d'abord ρ_1 car elle est au-dessus, puis ρ_2 - et les éventuelles suivantes dans le même ordre). L'effet de ρ_1 sur une permutation π sera par définition d'inverser l'ordre des éléments appartenant à l'intervalle qu'elle couvre ainsi que leur signe ; si on avait $\rho_2 = \rho_1$, on reviendrait au point de départ (l'identité), mais comme ici ρ_2 dépasse, elle va non seulement annuler l'effet de ρ_1 mais également effectuer sur cet intervalle un glissement et remplacer les $l_2 - l_1$ premières positions par l'inversion de l'intervalle non couvert par ρ_1.
   Ayant la permutation $\pi = (\pi_0\ \pi_1\ \cdots\ \pi_i\ \pi_{i+1}\ \cdots\ \pi_{n-1})$, l'application de $\rho_1(0, i)$ (on peut bien sûr choisir d'autres indices, le 0 a été choisi pour simplifier l'exposé) donnera :

$$\sigma = (-\pi_i\ -\pi_{i-1}\ \cdots\ -\pi_1\ -\pi_0\ \pi_{i+1}\ \cdots\ \pi_j\ \pi_{j+1}\ \cdots\ \pi_{n-1})$$

   et l'application de $\rho_2(0, j)$ avec $j > i$ sur cette dernière donnera la permutation :

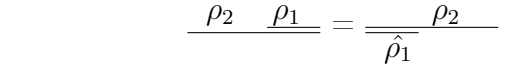
$$\tau = (-\pi_j\ -\pi_{j-1}\ \cdots\ -\pi_{i+2}\ -\pi_{i+1}\ \pi_0\ \pi_1\ \cdots\ \pi_i\ \pi_{j+1}\ \pi_{j+2}\ \cdots\ \pi_{n-1})$$

   Mais l'on peut aisément revenir à π, en inversant l'intervalle allant de π_0 à π_i et en remettant ensuite l'intervalle allant de π_j à π_0 dans le bon ordre et avec le bon signe à l'aide de l'inversion ρ_2. Si l'on applique ces opérations dans l'ordre inverse, on se rend compte que l'on a appliqué ρ_2 puis $\hat{\rho_1}$, qui est le symétrique de ρ_1 par rapport à ρ_2 ; graphiquement, on a donc :



   Et on a donc bien $(\rho_2 \circ \rho_1)\pi = (\hat{\rho_1} \circ \rho_2)\pi$; le cas ρ_2 préfixe de ρ_1 se traite de la même manière et l'on trouve $(\rho_2 \circ \rho_1)\pi = (\rho_1 \circ \hat{\rho_2})\pi$.

3. l'une est suffixe de l'autre : il suffit de prendre le raisonnement du "cas préfixe" dans l'autre sens et l'on trouve que $(\rho_2 \circ \rho_1)\pi = (\hat{\rho_1} \circ \rho_2)\pi$ si ρ_1 est suffixe de ρ_2, et $(\rho_2 \circ \rho_1)\pi = (\rho_1 \circ \hat{\rho_2})\pi$ si ρ_2 est suffixe de ρ_1. On a donc graphiquement le résultat :



Dans tous les cas décrits ci-dessus, l'unicité du couple (ρ_3, ρ_4) peut être montrée comme suit. Supposons qu'il existe un couple $(\rho'_3, \rho'_4) \neq (\rho_1, \rho_2) \neq (\rho_3, \rho_4)$ qui satisfait également $\rho_2 \circ \rho_1 = \rho'_4 \circ \rho'_3$. De toute évidence, les inversions ρ'_3 et ρ'_4 doivent être contenues dans le même intervalle qui contient les deux autres couples et elles doivent recouvrir à elles deux tout cet intervalle : ρ'_3 et ρ'_4 ne seraient donc pas liées et dans les cas préfixe/suffixe (le cas des inversions consécutives étant trivial), il devrait forcément s'agir d'inversions se chevauchant sinon l'une des deux serait égale à ρ_2 ou ρ_3. Mais deux inversions se chevauchant ne peuvent pas faire glisser un intervalle dans son intégralité comme le font les deux autres couples d'inversions ; donc (ρ'_3, ρ'_4) ne donne pas le même résultat et (ρ_3, ρ_4) est bien unique. □

Corollaire 5.1 $\underline{\underline{\rho_1 \quad \rho_2}}_{\rho_3} = \underline{\underline{\rho_1 \quad \hat{\rho}_3}}_{\rho_2} = \underline{\underline{\rho_2 \quad \hat{\rho}_3}}_{\hat{\rho}_1} = \underline{\underline{\quad \rho_2 \quad}}_{\rho_3 \quad \hat{\rho}_1}$

Le lemme suivant est similaire au lemme 5.1 mais s'applique à trois inversions.

Lemme 5.2 *Soit deux inversions $\rho_1(i_1, j_1)$ et $\rho_2(i_2, j_2)$, avec $i_2 < i_1 < j_1 < j_2$; alors la conjuguée de ρ_1 par ρ_2 coïncide avec $\hat{\rho}_1$ (le symétrique de ρ_1 par rapport à ρ_2) :*

$$\rho_2 \circ \rho_1 \circ \rho_2^{-1} = \hat{\rho}_1$$

(rappelons que $\rho_2 = \rho_2^{-1}$).

Démonstration : soit les inversions $\rho_1(i_1, j_1)$ et $\rho_2(i_2, j_2)$, avec $i_2 < i_1 < j_1 < j_2$; appliquons $\rho_2 \circ \rho_1 \circ \rho_2^{-1} = \rho_2 \circ \rho_1 \circ \rho_2$ à la permutation

$$\pi = (\pi_0 \ \pi_1 \ \cdots \ \pi_i \ \pi_{i+1} \ \cdots \ \pi_{n-1})$$

Afin de bien comprendre ce qui va suivre, il faut remarquer que $\rho_1(i_1, j_1)$ ne va pas agir sur les éléments $\pi_{i_1} \ \cdots \ \pi_{j_1}$, puisque ρ_2 aura été appliquée auparavant ; les éléments qui vont prendre la place de $\pi_{i_1} \ \cdots \ \pi_{j_1}$ seront donc $-\pi_{i_2+j_2-j_1} \ \cdots -\pi_{i_2+j_2-i_1}$ (se référer à la définition d'une inversion).

On a donc :

$$\begin{aligned}
(\rho_2 \circ \rho_1 \circ \rho_2)\pi &= (\rho_2 \circ \rho_1 \circ \rho_2)(\pi_0 \ \pi_1 \ \cdots \ \pi_{i_2} \ \cdots \ \pi_{i_1} \ \cdots \ \pi_{j_1} \ \cdots \ \pi_{j_2} \ \cdots \ \pi_{n-1}) \\
&= (\rho_2 \circ \rho_1)(\pi_0 \ \pi_1 \ \cdots \ -\pi_{j_2} \ \cdots \ -\pi_{i_2+j_2-j_1} \\
&\quad \cdots \ -\pi_{i_2+j_2-i_1} \ \cdots \ -\pi_{i_2} \ \cdots \ \pi_{n-1}) \\
&= \rho_2(\pi_0 \ \pi_1 \ \cdots \ -\pi_{j_2} \ \cdots \ \pi_{i_2+j_2-i_1} \ \cdots \ \pi_{i_2+j_2-j_1} \ \cdots \ -\pi_{i_2} \ \cdots \ \pi_{n-1}) \\
&= (\pi_0 \ \pi_1 \ \cdots \ \pi_{i_2} \ \cdots \ -\pi_{i_2+j_2-j_1} \ \cdots \ -\pi_{i_2+j_2-i_1} \ \cdots \ \pi_{j_2} \ \cdots \ \pi_{n-1})
\end{aligned}$$

Et on peut obtenir cette dernière permutation en appliquant $\hat{\rho}_1(i_2 + j_2 - j_1, i_2 + j_2 - i_1)$ à π :

$$\begin{aligned}
\hat{\rho}_1\pi &= \hat{\rho}_1(\pi_0 \ \pi_1 \ \cdots \ \pi_{i_2} \ \cdots \ \pi_{i_2+j_2-i_1} \ \cdots \ \pi_{i_2+j_2-j_1} \ \cdots \ \pi_{j_2} \ \cdots \ \pi_{n-1}) \\
&= (\pi_0 \ \pi_1 \ \cdots \ \pi_{i_2} \ \cdots \ -\pi_{i_2+j_2-j_1} \ \cdots \ -\pi_{i_2+j_2-i_1} \ \cdots \ \pi_{j_2} \ \cdots \ \pi_{n-1})
\end{aligned}$$

□

Remarque : si la notion de symétrique d'une inversion par rapport à une autre est aisément visualisable dans le cas des inversions liées, elle peut en revanche sembler quelque peu ambiguë dans le cas des inversions se chevauchant ; dans ce cas, l'intervalle utilisé pour construire les symétriques n'est pas celui couvert par une des inversions, mais l'intervalle $[\min_k(i_{\rho_k}), \max_k(j_{\rho_k})]$.

Avant d'établir la preuve du lemme suivant, examinons comment l'application de trois inversions à une permutation positive peut donner une autre permutation positive à distance 3 de la première ; pour ce faire, il faut que les deux premières inversions créent un et un seul intervalle d'éléments négatifs, que la troisième inversion se chargera de rendre positifs. Cette dernière est donc bien déterminée par les deux premières, qui peuvent soit être liées soit incluses l'une dans l'autre (comme dans le lemme 5.2).

Je vais maintenant prouver que $\rho_3 \circ \rho_2 \circ \rho_1 = \hat{\rho_1} \circ \hat{\rho_2} \circ \hat{\rho_3}$; d'après ce que je viens de décrire, on a $\rho_1(i_1, j_1)$, $\rho_2(i_2, j_2)$ et $\rho_3(i_3, j_3)$ avec $i_1 < i_2 < j_1 < j_2$, $i_3 = i_1$ et $j_3 = j_1 - i_2 + j_2$. L'intervalle utilisé pour construire les symétriques sera donc $I = [i_1, j_2]$ et l'on trouvera donc les symétriques $\hat{\rho_1}(i_1 + j_2 - j_1, j_2)$, $\rho_2(i_1, i_1 + j_2 - i_2)$ et $\rho_3(i_1 + j_2 - j_3, j_2)$.

Lemme 5.3 *A tout triple d'inversions appliqué à une permutation π correspond un et un seul autre triple d'inversions qui coïncide avec le premier :*

$$\rho_3 \circ \rho_2 \circ \rho_1 = \hat{\rho_1} \circ \hat{\rho_2} \circ \hat{\rho_3}$$

Démonstration : dans un souci de clarté, nous n'allons examiner ici que le cas où π est positive (la démonstration est similaire dans le cas général, et seul le cas positif sera utilisé dans la suite). Il suffit d'examiner ici le cas où les deux premières inversions se chevauchent, les autres cas ayant déjà été traités (corollaire 5.1). Nous allons, toujours par souci de clarté, utiliser une représentation graphique de la permutation et des inversions ; sur la figure 5.2, la permutation π est représentée en noir sous la forme d'un escalier dont chaque marche correspond à un élément. Cette représentation permet de distinguer les éléments, mais la forme croissante ne doit pas abuser le lecteur : il ne s'agit pas de la permutation identité, mais de n'importe quelle permutation positive ; la forme croissante sert justement à représenter les éléments positifs et lorsqu'une inversion sera appliquée, on aura une permutation comportant une section décroissante correspondant à des éléments négatifs. J'ai également représenté sous cette permutation les trois inversions ρ_1, ρ_2 et ρ_3 avec leurs bornes i_k, j_k, ainsi que l'ordre dans lequel elles sont appliquées (même convention graphique que précédemment) ; le trait en pointillés représente l'intervalle I, et sous cet intervalle se trouvent respectivement $\hat{\rho_1}$, $\hat{\rho_2}$ et $\hat{\rho_3}$ (les indices n'ont pas été représentés pour éviter de surcharger le dessin).

Appliquons maintenant $\rho_3 \circ \rho_2 \circ \rho_1$ à π : les différentes étapes de cette transformation sont montrées à la figure 5.3.

Nous appliquons également $\hat{\rho_1} \circ \hat{\rho_2} \circ \hat{\rho_3}$ à π : les différentes étapes de cette transformation sont montrées à la figure 5.4.

On a donc bien $(\rho_3 \circ \rho_2 \circ \rho_1)\pi = (\hat{\rho_1} \circ \hat{\rho_2} \circ \hat{\rho_3})\pi$, puisqu'on a exactement la même permutation comme résultat. La démonstration dans le cas où ρ_2 chevauche ρ_1 "par la gauche", c'est-à-dire où l'on n'a plus $i_1 < i_2 < j_1 < j_2$ mais $i_2 < i_1 < j_2 < j_1$, est

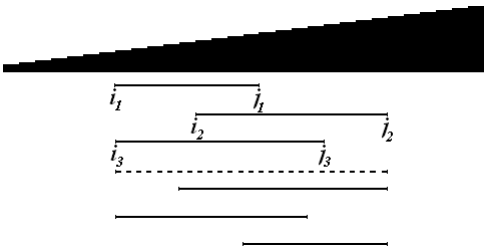


FIG. 5.2 – La permutation π et les intervalles couverts par ρ_1, ρ_2, ρ_3, $\hat{\rho}_1$, $\hat{\rho}_2$, $\hat{\rho}_3$

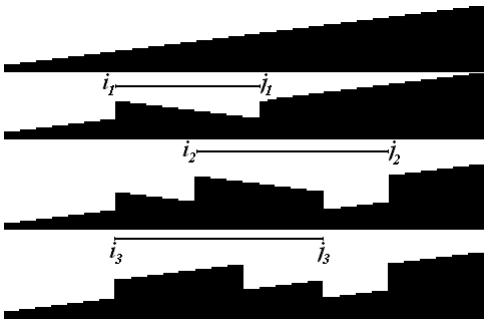


FIG. 5.3 – $(\rho_3 \circ \rho_2 \circ \rho_1)\pi$

similaire. L'unicité du triple peut être aisément montrée par un argument similaire à celui utilisé au lemme 5.1. □

Enfin, je vais montrer que si l'on rajoute des éléments en bonne place dans une permutation, cela ne modifie pas sa distance.

Lemme 5.4 *Si la distance de la permutation encadrée* $\pi = (0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n+1)$ *est* $d(\pi)$*, alors la distance des permutations contenant* π *et qui sont de la forme* $\sigma = (0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n+1\ n+2\ \cdots\ n+k)$*, est également de* $d(\pi)$*.*

Démonstration : si l'on ajoute à la permutation $\pi = (0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n{+}1)$ l'élément $n+2$, la transformant ainsi en la permutation $\sigma = (0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n+1\ \ n+2)$, le nombre d'éléments augmente de 1, mais également le nombre de cycles, comme le montre la figure 5.5 (les arcs de π n'ont pas été représentés par souci de clarté et de généralité).

Comme la nouvelle composante n'a pas de lien avec celles du graphe de π, ce n'est pas une haie ; le nombre de haies n'augmente donc pas, et on ne change donc

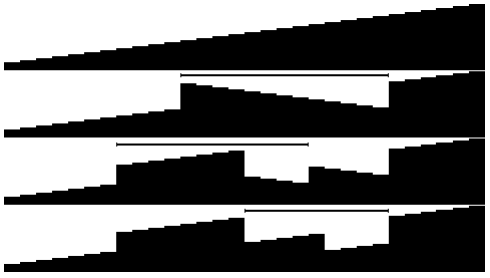


FIG. 5.4 – $(\hat{\rho}_1 \circ \hat{\rho}_2 \circ \hat{\rho}_3)\pi$

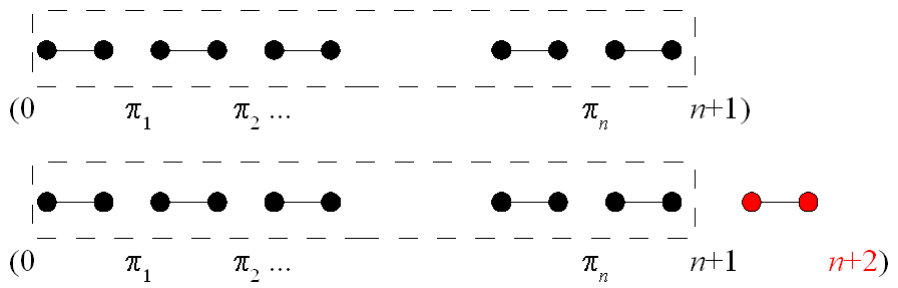


FIG. 5.5 – Graphes des cycles respectivement de $(0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n+1)$ et $(0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n+1\ n+2)$

pas l'état de forteresse de la permutation (si π était une forteresse, alors σ aussi, et si π n'était pas une forteresse, alors σ non plus). La distance de σ est donc bien égale à $d(\pi)$. □

Mes résultats précédents, à l'exception de la proposition 5.1, vont me servir à énumérer les permutations positives à distance 3 dans la proposition suivante.

Proposition 5.2 $\forall n \geq 3 : \left[\!\left[{n \atop 3}\right]\!\right]^+ = \binom{n}{4}$

Démonstration : La preuve consiste en une énumération des combinaisons de 3 inversions sur l'identité donnant une permutation positive encadrée à distance 3, en s'appuyant sur le fait que certaines combinaisons d'inversions sont équivalentes. On va procéder par induction, en se basant sur $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

1. Cas de base : $n = 3$: la seule permutation positive est l'identité et elle ne convient pas ici puisque sa distance est 0 ; il y a donc bien $0 = \binom{3}{4}$ permutations positives à distance 3.
2. Induction : vrai pour $n-1 \Rightarrow$ vrai pour n : on peut constater que les permutations d'ordre n contiennent (au sens du lemme 5.4) toutes les permutations d'ordre $n-1$, en particulier celles qui nous intéressent, les permutations à distance 3 ; on a donc déjà au moins $\binom{n-1}{4}$ permutations de n éléments à distance 3 (cf. lemme 5.4). Remarquons que l'énumération des composées de trois inversions donnant une permutation positive à distance 3 se ramène à une énumération de composées de deux inversions $\rho_1(i_1, j_1)$ et $\rho_2(i_2, j_2)$, car la troisième sera automatiquement déterminée de manière unique par les deux précédentes : en effet, cette dernière inversion doit rendre positifs tous les éléments rendus négatifs par les deux premières inversions. Parmi les couples d'inversions possibles, on écartera donc le cas des inversions non contiguës (donc agissant sur deux intervalles séparés par un nombre non nul d'éléments), puisqu'il sera impossible de rendre ces éléments positifs en une seule inversion.
   Caractérisons les permutations restantes : ce sont celles qui impliquent au moins une inversion manipulant l'élément $n-2$ (auquel on n'a pas pu toucher à l'étape précédente), donc sur l'intervalle $[k \overset{\pi}{—} n-2]$ pour k variant entre 1 et $n-2$. Remarquons qu'il ne faut pas compter les couples d'inversions dont l'une est préfixe ou suffixe de l'autre, car par le corollaire 5.1 on peut les obtenir

à partir des couples d'inversions consécutives que l'on aura énumérés. Nous allons donc compter :

(a) les couples d'inversions consécutives dont l'une touche à l'élément $n-2$: pour calculer ce nombre, il suffit de prendre toutes les permutations de $n-1$ éléments à distance 1 et d'appliquer à chacune d'entre elles une inversion $\rho_2(i_2, j_2)$ avec $i_2 = j_1 + 1$ et $j_2 = n-2$, ce qui revient en fait à compter le nombre de permutations de $n-1$ éléments à distance 1 puisque dans ce cas précis, ρ_2 est déterminée de manière unique par ρ_1 ; il y a donc $\left[\!\left[{n-1 \atop 1} \right]\!\right] = \binom{n-2}{2}$ (cf. proposition 5.3) permutations à distance 3 que l'on peut obtenir de cette manière.

(b) les couples d'inversions se chevauchant telles que l'une touche à l'élément $n-2$: il nous suffit dans ce cas d'énumérer les couples d'inversions tels que seule ρ_1 agit sur $n-1$ (cf. lemme 5.3). Dans ce cas, on a $2 \leq l_1 \leq n-3$, et $\rho_2(i_2, j_2)$ doit être telle que j_2 appartient à l'intervalle couvert par ρ_1 (donc il y a $l_1 - 1$ choix pour j_2) et i_2 n'appartienne pas à cet intervalle (donc il y a $n - 2 - l_1$ choix pour i_2). On a donc $\sum_{i=2}^{n-3}(i-1)(n-2-i)$ permutations obtenues de cette façon.

$$\overbrace{\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet\bullet}^{n-2}\overbrace{\qquad\qquad}^{l_1}$$

Or, en posant $j = i - 1$:

$$\begin{aligned}
\sum_{i=2}^{n-3}(i-1)(n-2-i) &= \sum_{j=1}^{n-4} j(n-3-j) \\
&= (n-3)\sum_{j=1}^{n-4} j - \sum_{j=1}^{n-4} j^2 \\
&= (n-3)\frac{(n-4)(n-3)}{2} - \frac{(n-4)(2(n-4)+1)(n-3)}{6} \\
&= \frac{(n-4)(n-3)}{6}(3(n-3) - (2n-7)) \\
&= \frac{(n-4)(n-3)}{6}(3n-9-2n+7) \\
&= \frac{(n-4)(n-3)(n-2)}{6} \\
&= \frac{(n-2)(n-3)(n-4)(n-5)!}{3!(n-5)!} \\
&= \binom{n-2}{3}
\end{aligned}$$

En ajoutant à ce résultat les inversions obtenues en (a) et celles que l'on possède par hypothèse d'induction, on a :

$$\begin{aligned}\left[\!\left[{n \atop 3} \right]\!\right]^{+} &= \binom{n-1}{4} + \binom{n-2}{2} + \binom{n-2}{3} \\ &= \binom{n-1}{4} + \binom{n-1}{3} \\ &= \binom{n}{4}\end{aligned}$$

□

La transformation de $\sum_{j=1}^{n} j^2$ est expliquée en long et en large de sept manières différentes dans [9].

5.3 Observations sur les permutations encadrées (non nécessairement positives)

Cette section présente les démonstrations relatives à quelques observations que l'on peut faire sur $\left[\!\left[{n \atop k} \right]\!\right]^{-}$ et $\left[\!\left[{n \atop k} \right]\!\right]$, sur base des tableaux 5.4 et 5.5.

Je donne dans la proposition suivante le nombre de permutations à distance 1.

Proposition 5.3 $\left[\!\left[{n \atop 1} \right]\!\right] = \binom{n-1}{2}$

Démonstration : les permutations encadrées à distance 1 sont triées, à part une séquence de $1 \leq t \leq n-2$ éléments décroissants et tous négatifs, ce qui fait qu'une seule inversion suffit non seulement à mettre les éléments de ladite séquence dans le bon ordre mais également à leur donner le bon signe (bien entendu, un tel intervalle ne peut exister que si $n \geq 3$). $\left[\!\left[{n \atop 1} \right]\!\right]$ est donc donné par le nombre de manières de choisir les bornes de l'inversion à appliquer :

- soit les deux bornes sont égales, et dans ce cas, l'inversion agit sur un seul élément : dans ce cas, il y a $\binom{n-2}{1}$ manières de choisir l'élément dont on va inverser le signe ($n-2$ car 0 et $n-1$ ne peuvent être choisis) ;
- soit les deux bornes sont différentes, et dans ce cas, il y a $\binom{n-2}{2}$ manières de choisir les deux bornes parmi les $n-2$ candidats acceptables.

On a donc $\left[\!\left[{n \atop 1} \right]\!\right] = \left[\!\left[{n \atop 1} \right]\!\right]^{-} = \binom{n-2}{1} + \binom{n-2}{2} = \binom{n-1}{2}$ (pour rappel, $\left[\!\left[{n \atop 1} \right]\!\right]^{+} = 0$). □

Dans la section précédente, nous nous étions intéressé à la permutation χ. Il existe d'autres permutations remarquables : ce sont celles qui seraient l'identité si l'on considérait leurs éléments en valeur absolue, mais qui en réalité comportent des éléments négatifs. Une telle permutation est par exemple :

$$(0\ -1\ 2\ 3\ -4\ -5\ 6)$$

qui est triée en trois inversions. Je qualifie ces permutations de *pseudo-identitaires* et je vais montrer qu'elles sont intéressantes par l'expression de leur distance : en effet, si p est le nombre d'éléments négatifs qu'une telle permutation comporte, alors p inversions de longueur 1 suffisent à la trier.

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$	$\binom{n}{7}$	$\binom{n}{8}$	$\binom{n}{9}$	$\binom{n}{10}$	$\binom{n}{11}$	$\binom{n}{12}$	$\binom{n}{13}$	$\binom{n}{14}$	$\binom{n}{15}$
0	1															
1	1	1														
2	1	2	1													
3	1	3	3	1												
4	1	4	6	4	1											
5	1	5	10	10	5	1										
6	1	6	15	20	15	6	1									
7	1	7	21	35	35	21	7	1								
8	1	8	28	56	70	56	28	8	1							
9	1	9	36	84	126	126	84	36	9	1						
10	1	10	45	120	210	252	210	120	45	10	1					
11	1	11	55	165	330	462	462	330	165	55	11	1				
12	1	12	66	220	495	792	924	792	495	220	66	12	1			
13	1	13	78	286	715	1287	1716	1716	1287	715	286	78	13	1		
14	1	14	91	364	1001	2002	3003	3432	3003	2002	1001	364	91	14	1	
15	1	15	105	455	1365	3003	5005	6435	6435	5005	3003	1365	455	105	15	1

Tab. 5.3 – Triangle de Pascal

Proposition 5.4 *Si π est une permutation pseudo-identitaire comportant p éléments négatifs, alors $d(\pi) = p$.*

Démonstration : remarquons d'abord qu'une permutation pseudo-identitaire ne comporte pas de haie ; en effet, considérons les différentes configurations possibles, montrées à la figure 5.6, représentant un morceau du graphe des cycles de π dans les cas respectifs où i, i et $i+1$, ou i ; $i+1$ et $i+2$ sont négatifs (le choix des éléments est arbitraire, et l'on aurait très bien pu choisir d'autres éléments négatifs, à gauche de i par exemple).

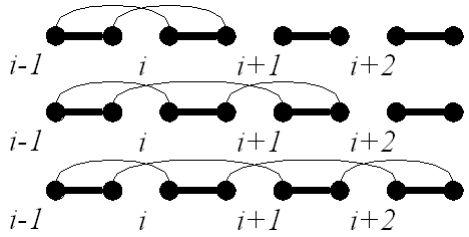


FIG. 5.6 – Trois graphes des cycles correspondant chacun à un morceau d'une permutation pseudo-identitaire ; sur le premier graphe , i est négatif, sur le deuxième, i et $i+1$ sont négatifs et sur le dernier, i, $i+1$ et $i+2$ sont négatifs

On peut constater que toute composante connexe de ce graphe des cycles possède au moins deux arêtes dont le support contient exactement trois éléments, donc ces arêtes sont par définition impaires ; ce qui implique également que toutes les composantes connexes du graphe de chevauchement (qu'il nous est inutile de représenter ici) seront orientées, et comme une haie est définie comme une composante non orientée du graphe de chevauchement, on peut en conclure que π n'en contient aucune, et n'est donc forcément pas non plus une forteresse. Sa distance peut alors se réécrire comme $d(\pi) = n - 1 - c$.

Le schéma d'inversions proposées, consistant à inverser un à un les signes des éléments négatifs, est bel et bien optimal ; en effet, l'application d'un simple changement de signe à un élément négatif a pour effet de créer un nouveau cycle, faisant décroître d'autant $d(\pi)$, ce qui est la définition d'une inversion sûre telle que décrite dans [3]. Comme chacune de ces inversions est sûre et que toute séquence d'inversions sûres est optimale [10], la thèse est bien vérifiée. □

Le décompte des permutations pseudo-identitaires fournit une minoration du nombre total de permutations :

Proposition 5.5 $\forall k > 0 : \left[\!\left[{n \atop k} \right]\!\right]^{-} \geq \binom{n-2}{k}$

Démonstration : une permutation pseudo-identitaire contient au moins un élément négatif et s'obtient à partir de l'identité en plaçant des signes $-$; seuls $n-2$ éléments peuvent devenir négatifs et pour obtenir une permutation pseudo-identitaire à distance k, il y a $\binom{n-2}{k}$ manières de placer k signes $-$. Le signe $\geq$ est justifié par le fait que l'on n'a traité ici que le cas des permutations pseudo-identitaires ; de plus, il faut $k > 0$ car la seule permutation à distance 1 est l'identité qui par définition

est positive. □

Pour clôturer cette section, je donne dans la proposition ci-dessous le nombre de permutations à distance 2.

Proposition 5.6 $\forall n \geq 3 : \left[\!\left[{n \atop 2} \right]\!\right] = \frac{(n-3)(n-2)(n-1)^2}{6}$

Démonstration : $\frac{n(n+1)(n+2)^2}{6} = \sum_{i=2,j<i}^{n}(i^2 - j^2)$ [20], qui par récurrence peut s'exprimer sous la forme

$$a(n) = a(n-1) + \sum_{i=1}^{n-1}(n^2 - i^2)$$

Nous allons donc logiquement procéder à une démonstration par induction.

1. Cas de base : $n = 3$: les permutations encadrées de trois éléments sont (0 1 2) et (0 −1 2), respectivement de distances 0 et 1. On a donc un nombre de permutations à distance 2 égal à

$$\left[\!\left[{3 \atop 2} \right]\!\right] = 0 = \frac{(3-3)(3-2)(3-1)^2}{6}$$

2. Induction : vrai pour $n-1 \Rightarrow$ vrai pour n : par hypothèse, on a déjà $a(n-4) = \frac{(n-4)(n-3)(n-2)^2}{6}$ permutations encadrées de n éléments à distance 2 (puisque le rajout d'éléments déjà en bonne place après le dernier ne modifie pas la distance, voir lemme 5.4). Maintenant qu'on a rajouté l'élément $n-1$ comme "cadre supérieur", il reste à énumérer les couples d'inversions dont :
   (a) la seconde seulement touche à l'élément $n-2$: $n-2$ inversions potentielles touchent à cet élément, et on les applique sur des permutations à distance 1 à l'origine de $n-1$ éléments (on n'aura donc jamais dans ce cas-ci un couple d'inversions égales) dont on connaît le nombre par la proposition 5.3. On peut donc de cette manière obtenir $\left[\!\left[{n-1 \atop 1} \right]\!\right](n-2) = \binom{n-2}{2}(n-2)$
   (b) la première seulement touche à l'élément $n-2$: si les deux inversions sont distinctes (elles n'ont aucun élément en commun) ou si la seconde est préfixe de la première, par symétrie (cf. lemme 5.1) on les a déjà comptées. Il ne reste donc plus qu'à énumérer les couples d'inversions tels que :
      i. ρ_2 est contenue strictement dans ρ_1 : il ne faut plus les compter non plus, car par symétrie on les a déjà (voir lemme 5.2)
      ii. ρ_2 chevauche ρ_1 de k éléments : comme on l'a vu dans la proposition 5.2, il y a $\sum_{i=1}^{n-2}(i-1)(n-2-i)$ couples (et donc permutations résultantes) possibles.
      iii. ρ_1 est suffixe de ρ_2 : sur un intervalle de longueur $n-2$, il y a $n-2$ inversions contenant l'élément $n-2$; les inversions de longueur inférieure à la valeur de l_1 courante ne peuvent forcément pas la contenir et il ne reste donc que les suivantes, au nombre de $n-2-l_1$. La longueur de ρ_1 pouvant varier entre 1 et $n-2$, on obtient donc $\sum_{i=1}^{n-2}(n-2-i)$ permutations de cette manière.

En rassemblant les deux sommes, on a :

$$\begin{aligned}
& \sum_{i=1}^{n-2}(i-1)(n-2-i)+\sum_{i=1}^{n-2}(n-2-i) \\
= & \sum_{i=1}^{n-2} i(n-2-i) \\
= & (n-2)\sum_{i=1}^{n-2} i-\sum_{i=1}^{n-2} i^2 \\
= & \frac{(n-2)^2(n-1)}{2}-\frac{(n-2)(2(n-2)+1)(n-1)}{6} \\
= & \frac{(n-2)(n-1)}{2}\left(n-2-\frac{2n-4+1}{3}\right) \\
= & \frac{(n-2)(n-1)}{2}\frac{3n-6-2n+3}{3} \\
= & \frac{(n-2)(n-1)}{2}\frac{(n-3)}{3}
\end{aligned}$$

Ajoutons-y les $\binom{n-2}{2}(n-2)$ autres permutations :

$$\begin{aligned}
& \binom{n-2}{2}(n-2)+\frac{(n-2)(n-1)}{2}\frac{(n-3)}{3} \\
= & \frac{(n-2)^2(n-3)}{2}+\frac{(n-2)(n-1)}{2}\frac{(n-3)}{3} \\
= & \frac{(n-2)(n-3)}{2}\left(n-2+\frac{(n-1)}{3}\right) \\
= & \frac{(n-2)(n-3)}{2}\left(\frac{3n-6+n-1}{3}\right) \\
= & \frac{(n-2)(n-3)}{2}\left(\frac{4n-7}{3}\right)
\end{aligned}$$

Et finalement, ajoutons à cette quantité les $a(n-4)$ permutations que l'on avait déjà par hypothèse d'induction :

$$\begin{aligned}
& a(n-4)+\frac{(n-2)(n-3)}{2}\left(\frac{4n-7}{3}\right) \\
= & \frac{(n-4)(n-3)(n-2)^2}{6}+\frac{(n-2)(n-3)}{2}\left(\frac{4n-7}{3}\right) \\
= & \frac{(n-3)(n-2)}{6}((n-4)(n-2)+4n-7) \\
= & \frac{(n-3)(n-2)}{6}(n^2-6n+8+4n-7)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{(n-3)(n-2)}{6}(n^2-2n+1) \\
&= \frac{(n-3)(n-2)(n-1)^2}{6} = a(n-3)
\end{aligned}
$$

□

Enfin, le tableau 5.6 fournit une mesure expérimentale du nombre maximal de haies dans une permutation (quelconque) ; on voit qu'il semble s'en dégager une relation, mais je n'ai pas réussi à la formuler plus clairement ni à la démontrer. Il semble d'ailleurs peut-être un peu prématuré de vouloir formuler une conjecture à ce sujet avec si peu de données.

n	$\left[\!\left[{n \atop 0} \right]\!\right]$	$\left[\!\left[{n \atop 1} \right]\!\right]$	$\left[\!\left[{n \atop 2} \right]\!\right]$	$\left[\!\left[{n \atop 3} \right]\!\right]$	$\left[\!\left[{n \atop 4} \right]\!\right]$	$\left[\!\left[{n \atop 5} \right]\!\right]$	$\left[\!\left[{n \atop 6} \right]\!\right]$	$\left[\!\left[{n \atop 7} \right]\!\right]$	$\left[\!\left[{n \atop 8} \right]\!\right]$	$\left[\!\left[{n \atop 9} \right]\!\right]$	$\left[\!\left[{n \atop 10} \right]\!\right]$
1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	0
4	1	3	3	1	0	0	0	0	0	0	0
5	1	6	16	25	0	0	0	0	0	0	0
6	1	10	50	173	142	8	0	0	0	0	0
7	1	15	120	721	1543	1437	3	0	0	0	0
8	1	21	245	2256	8835	19503	15039	180	0	0	0
9	1	28	448	5851	35909	139066	261936	201800	81	0	0
10	1	36	756	13279	116838	691818	2296671	4194607	2999662	8252	0
11	1	45	1200	27266	324430	2708878	13740615	43527112	73879601	51534622	50790

TAB. 5.4 – Quelques valeurs expérimentales de $\left[\!\left[{n \atop k} \right]\!\right]$

n	$\left[\!\left[{n \atop 0} \right]\!\right]^-$	$\left[\!\left[{n \atop 1} \right]\!\right]^-$	$\left[\!\left[{n \atop 2} \right]\!\right]^-$	$\left[\!\left[{n \atop 3} \right]\!\right]^-$	$\left[\!\left[{n \atop 4} \right]\!\right]^-$	$\left[\!\left[{n \atop 5} \right]\!\right]^-$	$\left[\!\left[{n \atop 6} \right]\!\right]^-$	$\left[\!\left[{n \atop 7} \right]\!\right]^-$	$\left[\!\left[{n \atop 8} \right]\!\right]^-$	$\left[\!\left[{n \atop 9} \right]\!\right]^-$
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
4	0	3	3	0	0	0	0	0	0	0
5	0	6	16	20	0	0	0	0	0	0
6	0	10	50	158	142	0	0	0	0	0
7	0	15	120	686	1543	1356	0	0	0	0
8	0	21	245	2186	8835	19058	15015	0	0	0
9	0	28	448	5725	35909	137305	261828	198820	17	0
10	0	36	756	13069	116838	686194	2296310	4169408	2998805	184

TAB. 5.5 – Quelques valeurs expérimentales de $\left[\!\left[{n \atop k} \right]\!\right]^-$

n	Nombre maximal de haies
3	0
4	1
5	1
6	1
7	2
8	2
9	2
10	3
11	3

TAB. 5.6 – Quelques valeurs du nombre maximal de haies dans une permutation encadrée de n éléments

Chapitre 6

Distances invariantes à droite et mesures de désordre

Les distances invariantes à droite (définition 6.1) sont des distances sur l'ensemble des permutations ; informellement, une distance invariante à droite est une distance telle que sa valeur n'est pas modifiée lors de l'application à droite de la même transformation aux deux éléments dont elle mesure l'écart. Quelques exemples seront cités dans ce chapitre, mais nous pouvons déjà évoquer celui de la distance des inversions étudiée dans les chapitres précédents.

Toujours informellement, une mesure de désordre est la distance d'une permutation à la permutation identique ; nous expliciterons une correspondance biunivoque entre mesures de désordre et distances invariantes à droite. Les distances invariantes à droite sont notamment utilisées pour mesurer le désordre existant dans une permutation (cf. Estivill-Castro, Mannila et Wood [6]). Par exemple, il semble assez intuitif que (1 2 3 5 4) est "moins désordonnée" que (5 4 1 3 2), en ce sens que la première permutation est plus "proche" de l'identité ι que la seconde. A titre d'exemple, la distance des inversions fournit une mesure de désordre et beaucoup d'autres exemples existent, dont quelques-uns seront présentés dans ce chapitre. Il paraît normal que plus une permutation est ordonnée, plus elle devrait être triée rapidement ; les mesures de désordre permettent justement de construire des algorithmes de tri dits *adaptatifs*, devant leur nom au fait que leur complexité dépend non seulement de la taille de la permutation à trier mais également du désordre (ou de l'ordre) présent dans cette dernière.

La théorie que nous venons d'évoquer concerne les permutations au sens classique, donc non signées ; l'extension au cas signé semble par contre avoir été moins étudiée (à l'exception notable de la théorie des groupes de Coxeter, où le cas signé est un exemple pour lequel des distances jouent un rôle important ; nous rencontrerons ces distances dans la dernière section). Comme il existe de nombreuses distances entre des permutations non signées, et que les distances invariantes à droite sont particulièrement intéressantes comme cela a été mentionné plus haut, nous avons voulu voir comment cette notion d'invariance à droite pourrait être adaptée et exploitée dans le cas des permutations signées.

6.1 Notations

Le problème du tri (informellement, ranger dans l'ordre croissant un ensemble ordonné d'éléments) est envisagé par Estivill-Castro et Wood [7] (et d'ailleurs plus généralement en informatique) dans le contexte des *séquences*, c'est-à-dire des ensembles ordonnés d'entiers non nécessairement tous distincts et non nécessairement tous consécutifs une fois la séquence triée (ce qui revient à dire qu'il peut y avoir des trous dans la numérotation). L'exposé de [7] sera appliqué ici aux permutations. Les séquences de [7] ne comportent pas d'éléments dupliqués, et il est toujours possible de renuméroter les éléments de toute séquence de manière à obtenir une permutation dans laquelle il ne manque pas d'élément (par exemple[1] : $\langle 6\ 1\ 18\ 5\ 613\ 32\rangle$ peut être renumérotée en $\langle 3\ 1\ 4\ 2\ 6\ 5\rangle$). Les définitions de [7] ont donc été adaptées ici aux permutations.

Dans la suite, π et σ désigneront comme au chapitre 3 des permutations des éléments 1, 2, ..., n. L'ensemble des $n!$ permutations de $\{1, 2, ..., n\}$ forme un groupe pour la composition, noté S_n. Le cardinal d'un ensemble A sera noté $|A|$. Le nombre d'éléments d'une permutation π désigne le nombre d'éléments de l'ensemble sur lequel agit π, c'est-à-dire le nombre de composantes dans le vecteur auquel s'identifie π (cf. chapitre 2) ; il sera noté $|\pi|$.

6.2 Définitions

Définition 6.1 *Une distance (définition 2.3)* $d : S_n \times S_n \to \mathbb{R}$ *est* invariante à droite *si*

$$d(\pi, \sigma) = d(\pi \circ \tau, \sigma \circ \tau),\ \forall\ \pi, \sigma, \tau \in S_n$$

Dans la suite, nous étudierons plusieurs familles de distances $(d_n)_{n\in\mathbb{N}}$ avec chaque distance d_n invariante à droite sur S_n. Les mesures de désordre que nous allons définir sont, comme nous le verrons, intimement liées aux distances invariantes à droite. Si d est invariante à droite sur S_n, elle est entièrement déterminée par

$$m : S_n \to \mathbb{R} : \pi \mapsto d(\pi, \iota) \tag{6.1}$$

car $d(\pi, \sigma) = d(\pi \circ \sigma^{-1}, \iota) = m(\pi \circ \sigma^{-1})$. Cette application m sera une mesure de désordre. La définition suivante exprime plus formellement cette notion.

Définition 6.2 *Soit* $m : S_n \to \mathbb{R}$ *; alors* m *est une* mesure de désordre *si et seulement si :*

1. $\forall\ \pi \in S_n : m(\pi) \geq 0$ *et* $m(\pi) = 0 \Leftrightarrow \pi = \iota$ *;*
2. $\forall\ \pi \in S_n : m(\pi) = m(\pi^{-1})$ *;*
3. $\forall\ \pi, \sigma \in S_n : m(\pi \circ \sigma) \leq m(\pi) + m(\sigma)$.

Si m est définie par l'équation (6.1) à partir de la distance invariante à droite d, ces axiomes sont directement dérivés de ceux de la définition 2.3 ; en effet :

[1] Afin d'éviter toute confusion, les séquences seront encadrées par des $\langle\ \rangle$, au lieu des parenthèses réservées aux permutations.

1. le caractère positif de m découle clairement de $d \geq 0$, et de plus $m(\pi) = 0$ donne $d(\pi, \iota) = 0$, donc $\pi = \iota$;
2. la propriété de symétrie et l'invariance à droite impliquent $m(\pi) = d(\pi, \iota) = d(\iota, \pi) = d(\pi^{-1}, \iota) = m(\pi^{-1})$;
3. enfin, l'inégalité triangulaire donne la troisième propriété car

$$\begin{aligned} m(\pi \circ \sigma) = d(\pi \circ \sigma, \iota) &= d(\pi, \sigma^{-1}) \\ &\leq d(\pi, \iota) + d(\iota, \sigma^{-1}) \\ &= d(\pi, \iota) + d(\sigma, \iota) \\ &= m(\pi) + m(\sigma) \end{aligned}$$

Réciproquement, toute mesure de désordre m sur S_n provient comme dans l'équation (6.1) d'une certaine distance à droite d sur S_n. Pour montrer cela, il suffit de prendre d définie par

$$d(\pi, \sigma) = m(\pi \circ \sigma^{-1}), \forall\ \pi, \sigma \in S_n$$

(les vérifications sont aussi aisées que celles qui viennent d'être données). En conclusion, l'équation (6.1) met en bijection les distances invariantes à droite avec les mesures de désordre.

La notion suivante sert à exprimer l'optimalité d'un algorithme de tri adaptatif par rapport à une mesure donnée ; soit $below(z, n, m)$ l'ensemble des permutations de n éléments dont le désordre, mesuré par m, est inférieur ou égal à z :

$$below(z, n, m) = \{\sigma \in S_n \mid m(\sigma) \leq z\}$$

Définition 6.3 *Soit une mesure de désordre m et un algorithme de tri S effectuant $T_S(\pi)$ comparaisons sur une permutation π ; S est* optimal par rapport à m, *ou de façon plus concise,* m-optimal, *si :*

$$\exists\ c > 0, \forall\ \pi \in S_n : T_S(\pi) \leq c \max\{|\pi|, \log |below(m(\pi), |\pi|, m)|\}$$

Définition 6.4 *Soit $m_1, m_2 : S_n \to \mathbb{R}$ deux mesures de désordre :*

1. *m_1 est* algorithmiquement meilleure *que m_2 si tout algorithme m_1-optimal est m_2-optimal, et cela est noté $m_1 \leq_{alg} m_2$;*
2. *m_1 est* algorithmiquement équivalente *à m_2 si $m_1 \leq_{alg} m_2$ et $m_2 \leq_{alg} m_1$, et cela est noté $m_1 =_{alg} m_2$.*

6.3 Approche groupale

Cette approche permet de présenter un grand nombre de mesures de désordre et nous sera utile dans la suite pour démontrer quelques propriétés.

Définition 6.5 *Un sous-ensemble S de S_n est une* partie génératrice *du groupe $S_n, \circ$ si toute permutation π de S_n est le produit d'éléments de $S \cup S^{-1}$, où S^{-1} est l'ensemble des inverses des éléments de S.*

Prenons les trois exemples suivants (nous expliquerons plus en détails Inv_2 et Exc à la section suivante) :

1. à Inv_2 est associé l'ensemble S des transpositions d'éléments adjacents : ces transpositions sont notées $(i, i+1)$, avec $1 \leq i \leq n-1$. Ceci correspond à une inversion, notée comme dans les chapitres précédents $\rho(i, i+1)$: en effet, la permutation π est transformée en $\pi \circ (i, i+1)$:

   $$(\pi_1\ \pi_2\ \cdots\ \pi_{i-1}\ \pi_i\ \pi_{i+1}\ \pi_{i+2}\ \cdots\ \pi_n)$$

   doit donc devenir

   $$(\pi_1\ \pi_2\ \cdots\ \pi_{i-1}\ \pi_{i+1}\ \pi_i\ \pi_{i+2}\ \cdots\ \pi_n)$$

   c'est-à-dire que

   $$\begin{pmatrix} 1 & 2 & \cdots & i-1 & i & i+1 & i+2 & \cdots & n \\ \pi_1 & \pi_2 & \cdots & \pi_{i-1} & \pi_i & \pi_{i+1} & \pi_{i+2} & \cdots & \pi_n \end{pmatrix}$$

   doit devenir

   $$\begin{pmatrix} 1 & 2 & \cdots & i-1 & i & i+1 & i+2 & \cdots & n \\ \pi_1 & \pi_2 & \cdots & \pi_{i-1} & \pi_{i+1} & \pi_i & \pi_{i+2} & \cdots & \pi_n \end{pmatrix}$$

   ce qui correspond bien à l'inversion $\rho(i, i+1)$. Ces transpositions consistent donc tout simplement à échanger les éléments en positions i et $i+1$.

2. à Exc est associé l'ensemble S de toutes les transpositions ; celles-ci sont notées (i, j), avec $1 \leq i < j \leq n$. Elles consistent à échanger les éléments en positions i et j.

3. à Inv, la distance des inversions[2], est associé l'ensemble S des inversions $\rho(i, j)$, avec $1 \leq i < j \leq n$. Notons qu'appliquer l'inversion $\rho(i, j)$ à la permutation π revient à transformer π en la permutation $\pi \circ (i, j) \circ (i+1, j-1) \circ (i+2, j-2) \circ \cdots \circ (i + \left\lfloor \frac{j-i-1}{2} \right\rfloor, j - \left\lfloor \frac{j-i-1}{2} \right\rfloor)$.

Dans chacun de nos trois exemples, l'ensemble S considéré satisfait $S = S^{-1}$ et est de plus une partie génératrice du groupe S_n : en effet, comme nous le verrons dans la section 6.4.1, toute permutation de S_n peut s'obtenir comme produit de transpositions d'éléments adjacents, donc l'ensemble S associé à Inv_2 est bien une partie génératrice. Les deux autres ensembles introduits pour Exc et Inv contenant encore plus d'éléments, ils sont aussi générateurs.

Définition 6.6 *Soit S une partie génératrice de S_n telle que $S = S^{-1}$; posons pour chaque $\pi \in S_n$:*

$$m(\pi) = \min\{k \mid \pi = s_1 \circ s_2 \circ \cdots \circ s_k \text{ avec } s_1, ..., s_k \in S\}$$

Nous allons maintenant montrer que cette définition fournit une mesure de désordre (définition 6.2).

[2]Insistons sur le fait que nous nous intéressons bien ici au cas non signé.

Proposition 6.1 *L'application m de la définition 6.6 est une mesure de désordre.*

Démonstration :

1. $\forall\ \pi \in S_n : m(\pi) \geq 0$ et $m(\pi) = 0 \Leftrightarrow \pi = \iota$; (trivial)
2. $\forall\ \pi \in S_n : m(\pi) = m(\pi^{-1})$; en effet :

$$\begin{array}{rcl} \pi = s_1 \circ \cdots \circ s_k \quad \Rightarrow \quad \pi^{-1} & = & s_k^{-1} \circ \cdots \circ s_1^{-1} \\ & = & s'_1 \circ \cdots \circ s'_k \end{array}$$

   où s'_1, ..., $s'_k \in S$ car $S = S^{-1}$. Donc π et π^{-1} s'expriment avec le même nombre minimum d'éléments de S.

3. $\forall\ \pi, \sigma \in S_n : m(\pi \circ \sigma) \leq m(\pi) + m(\sigma)$; en effet, si $\pi = s_1 \circ \cdots \circ s_k$ et $\sigma = s'_1 \circ \cdots \circ s'_l$, alors $\pi \circ \sigma = s_1 \circ \cdots \circ s_k \circ s'_1 \circ \cdots \circ s'_l$ nécessitant donc au plus $k + l$ générateurs pris dans S. □

6.4 Quelques mesures de désordre

Définition 6.7 *Un couple d'éléments (π_i, π_j), avec $i \neq j$, dans une permutation π est* inversé *si $i < j$ et $\pi_i > \pi_j$.*

J'utiliserai la permutation π^1 = (7 3 2 4 9 5 8 1 6 10) comme exemple pour illustrer quelques-unes des mesures présentées ci-dessous. Les couples inversés de cette permutation sont les 17 couples (7, 3), (7, 2), (7, 4), (7, 5), (7, 1), (7, 6), (3, 2), (3, 1), (2, 1), (4, 1), (9, 5), (9, 8), (9, 1), (9, 6), (5, 1), (8, 1), (8, 6).

Nous recourrons dans la suite à la même notation pour désigner une mesure de désordre et la distance invariante à droite correspondante (bien que le nombre d'arguments de la fonction soit un dans le premier cas et deux dans le second).

6.4.1 Inv_2

Inv_2 est le nombre de couples inversés dans la permutation ; montrons que c'est aussi le nombre minimal d'inversions de longueur 2 requises pour trier π. En effet, pour trier π, nous pouvons procéder de la façon suivante. Commençons par trier la permutation par la fin, c'est-à-dire que nous allons commencer par placer n en bonne place : ceci nécessite un nombre d'inversions égal au nombre d'éléments avec lesquels n forme un couple inversé. Une fois n en place, il ne forme plus de couple inversé avec aucun élément : il faut alors continuer avec $n - 1$, qui nécessite un nombre d'inversions égal au nombre de couples inversés faisant intervenir $n - 1$, et ainsi de suite jusqu'à ce que tous les éléments soient en bonne place. Ceci montre que le nombre minimum requis d'inversions de longueur 2 pour trier π est inférieur ou égal au nombre de couples inversés de π ; en fait, nous avons ici une égalité car une inversion de longueur 2 ne peut éliminer qu'un seul couple inversé à la fois, car les autres éléments ne bougent pas.

Inv_2 ressemble à la distance des inversions dont nous avons parlé dans le chapitre 3, mais il ne s'agit pas de la même chose, car seuls *deux* éléments adjacents peuvent

être échangés. L'exemple choisi comporte 17 couples inversés (comme nous l'avons vu en section 6.4) et nous donne donc $Inv_2(\pi^1) = 17$; une séquence de 17 échanges d'éléments adjacents triant π^1 est montrée au tableau 6.1.

$(7\ \underline{3\ 2}\ 4\ 9\ 5\ 8\ 1\ 6\ 10)$
$(7\ 2\ 3\ 4\ \underline{9\ 5}\ 8\ 1\ 6\ 10)$
$(7\ 2\ 3\ 4\ 5\ \underline{9\ 8}\ 1\ 6\ 10)$
$(\underline{7\ 2}\ 3\ 4\ 5\ 8\ 9\ 1\ 6\ 10)$
$(2\ \underline{7\ 3}\ 4\ 5\ 8\ 9\ 1\ 6\ 10)$
$(2\ 3\ \underline{7\ 4}\ 5\ 8\ 9\ 1\ 6\ 10)$
$(2\ 3\ 4\ \underline{7\ 5}\ 8\ 9\ 1\ 6\ 10)$
$(2\ 3\ 4\ 5\ 7\ 8\ \underline{9\ 1}\ 6\ 10)$
$(2\ 3\ 4\ 5\ 7\ \underline{8\ 1}\ 9\ 6\ 10)$
$(2\ 3\ 4\ 5\ \underline{7\ 1}\ 8\ 9\ 6\ 10)$
$(2\ 3\ 4\ \underline{5\ 1}\ 7\ 8\ 9\ 6\ 10)$
$(2\ 3\ \underline{4\ 1}\ 5\ 7\ 8\ 9\ 6\ 10)$
$(2\ \underline{3\ 1}\ 4\ 5\ 7\ 8\ 9\ 6\ 10)$
$(\underline{2\ 1}\ 3\ 4\ 5\ 7\ 8\ 9\ 6\ 10)$
$(1\ 2\ 3\ 4\ 5\ 7\ 8\ \underline{9\ 6}\ 10)$
$(1\ 2\ 3\ 4\ 5\ 7\ \underline{8\ 6}\ 9\ 10)$
$(1\ 2\ 3\ 4\ 5\ \underline{7\ 6}\ 8\ 9\ 10)$
$(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10)$

TAB. 6.1 – Le tri d'une permutation par inversions de longueur 2

La distance Inv_2 entre deux permutations peut également être calculée à l'aide d'un dessin, comme l'illustre la figure 6.1 : il s'agit d'écrire les deux permutations l'une en-dessous de l'autre, leurs éléments alignés, puis de relier tous les éléments des deux permutations de manière à joindre la position d'un élément dans l'une à sa position dans l'autre ; Inv_2 est alors donné par le nombre d'intersections des segments ainsi tracés (il faut bien prendre garde à ne pas créer d'intersections multiples).

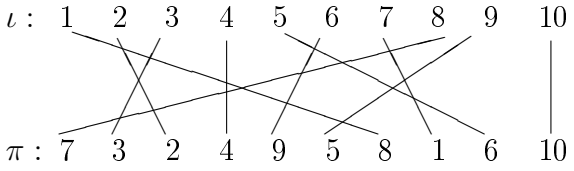


FIG. 6.1 – Une autre manière graphique pour le calcul de Inv_2

La figure 6.2, extraite de Knuth [15], montre le *permutoèdre* construit sur les 24 permutations de $\{1, 2, 3, 4\}$: il s'agit d'un polyèdre dont les sommets sont les permutations et dans lequel deux permutations π, σ sont jointes par une arête si $Inv_2(\pi, \sigma) = 1$. De manière graphique, la distance Inv_2 entre deux permutations de ce polyèdre s'obtient en cherchant le chemin minimal entre les deux sommets correspondants.

Ce permutoèdre peut aussi se représenté projeté dans le plan, ce qui est illustré à la figure 6.3.

Inv_2 est bien une mesure de désordre, car :

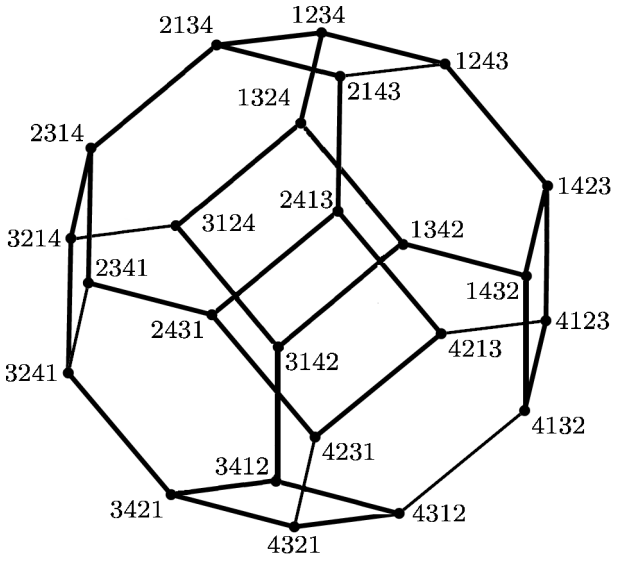


FIG. 6.2 – Le permutoèdre construit sur les permutations de {1, 2, 3, 4} (source : [15])

1. $\forall\ \pi \in S_n : Inv_2(\pi) \geq 0$ et $Inv_2(\pi) = 0 \Leftrightarrow \pi = \iota$ (trivial)
2. $\forall\ \pi \in S_n : Inv_2(\pi) = Inv_2(\pi^{-1})$. Une démonstration, issue de [15], s'effectue de la façon suivante : il s'agit de construire un tableau M $n \times n$, telle que $M_{ij} = \bullet$ si $\pi_i = j$, puis de marquer les cases M_{ij} qui sont restées vides d'une $\times$ si une des lignes $i+1$, ..., n de la colonne j contient un $\bullet$ et qu'une des colonnes $j+1$, ..., n de la ligne i contient également un $\bullet$. Le nombre de $\times$ est $Inv_2(\pi)$, car $M_{ij} = \times \Leftrightarrow (\pi_i, \pi_j)$ est un couple inversé ; la transposition de ce tableau, en échangeant les lignes et les colonnes, donne le tableau correspondant à π^{-1}, contenant le même nombre de $\times$ donc $Inv_2(\pi) = Inv_2(\pi^{-1})$. Le tableau correspondant à π^1 est donné en exemple à la table 6.2.

	1	2	3	4	5	6	7	8	9	10
1	×	×	×	×	×	×	•			
2	×	×	•							
3	×	•								
4	×			•						
5	×				×	×		×	•	
6	×				•					
7	×					×		•		
8	•									
9						•				
10										•

TAB. 6.2 – Le tableau correspondant à la permutation (7 3 2 4 9 5 8 1 6 10), illustrant la preuve de $Inv_2(\pi) = Inv_2(\pi^{-1})$

3. $\forall\ \pi, \sigma \in S_n : Inv_2(\pi \circ \sigma) \leq Inv_2(\pi) + Inv_2(\sigma)$: à chaque couple inversé de $\pi \circ \sigma$, nous allons associer soit un couple inversé de π, soit un couple inversé de

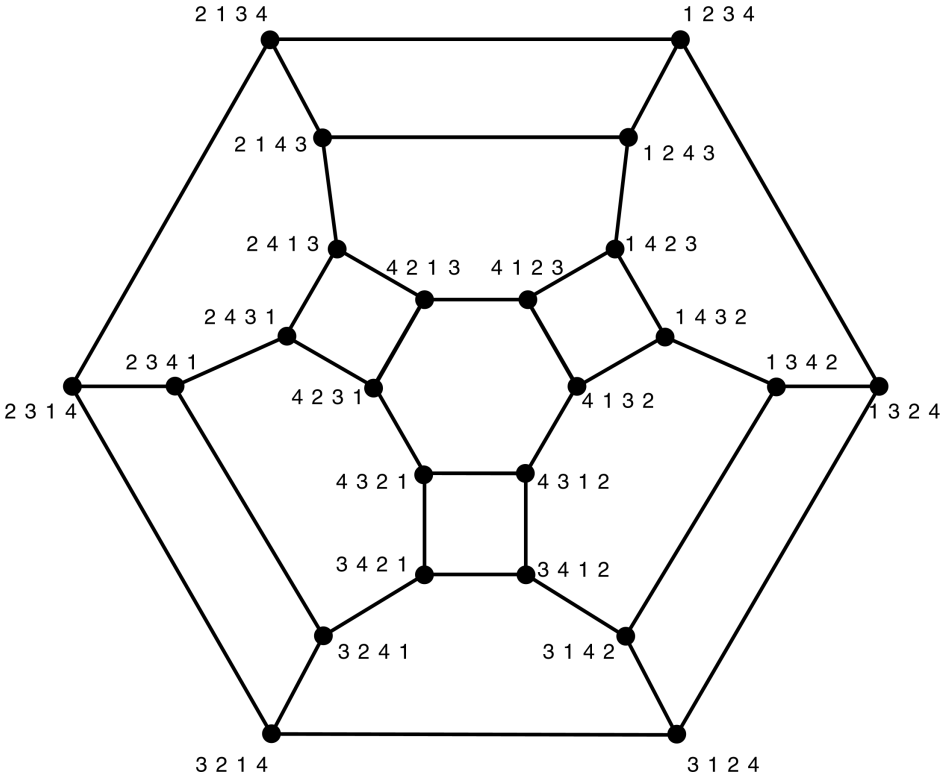


FIG. 6.3 – La projection du permutoèdre de la figure 6.2

σ. Donnons-nous un couple inversé $((\pi \circ \sigma)_i, (\pi \circ \sigma)_j)$ dans $\pi \circ \sigma$ (c'est-à-dire $i < j$ et $(\pi \circ \sigma)_i > (\pi \circ \sigma)_j$). Les deux cas possibles sont montrés à la figure 6.4 : soit σ est responsable de la création du couple inversé de $\pi \circ \sigma$, soit π l'est.

D'après la définition du produit de permutations, nous avons $(\pi \circ \sigma)_i = \pi_{\sigma_i}$ et $(\pi \circ \sigma)_j = \pi_{\sigma_j}$. Examinons les deux cas :

(a) $\sigma_i < \sigma_j$: alors $(\pi_{\sigma_j}, \pi_{\sigma_i})$ est un couple inversé dans π. Nous l'associons au couple inversé donné.

(b) $\sigma_i > \sigma_j$: alors (σ_i, σ_j) est un couple inversé dans σ, que nous associons au couple inversé donné.

Il est facile de voir que les couples inversés obtenus dans π sont tous différents, de même que les couples inversés obtenus dans σ. La formule $m(\pi \circ \sigma) \leq m(\pi) + m(\sigma)$ en résulte directement.

Pour montrer que Inv_2 est bien une mesure de désordre, nous aurions également pu utiliser la section 6.3 : la transformation de

$$\pi = (\pi_1\ \pi_2\ \cdots\ \pi_{i-1}\ \pi_i\ \pi_{i+1}\ \pi_{i+2}\ \cdots\ \pi_n)$$

en

$$\sigma = (\pi_1\ \pi_2\ \cdots\ \pi_{i-1}\ \pi_{i+1}\ \pi_i\ \pi_{i+2}\ \cdots\ \pi_n)$$

est, comme nous l'avons vu, l'échange $(i, i+1)$. Or, $Inv_2(\pi)$ est le plus petit nombre de telles opérations pour transformer π en ι : cela revient à trouver le plus petit k tel que :

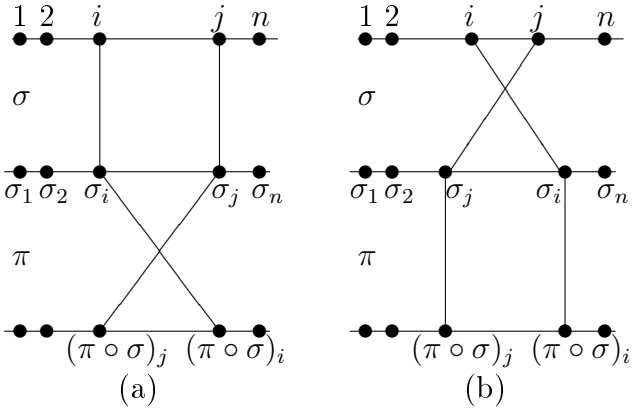


FIG. 6.4 – Les deux façons de créer un couple inversé dans $\pi \circ \sigma$

$$\begin{aligned} \pi \circ s_1 \circ s_2 \circ \cdots \circ s_k &= \iota \text{ avec } s_1, ..., s_k \text{ de la forme } (i, i+1) \\ \Leftrightarrow \pi &= (s_k)^{-1} \circ (s_{k-1})^{-1} \circ \cdots \circ (s_1)^{-1} \\ &= s_k \circ s_{k-1} \circ \cdots \circ s_1 \end{aligned}$$

Le fait que Inv_2 est une mesure de désordre découle donc aussi directement de la section 6.3.

6.4.2 *Max*

Max est la plus grande distance qu'un élément doit parcourir pour arriver dans la position qu'il occuperait une fois la permutation triée ; plus formellement, posons :

$$Max(\pi) = \max_{1 \leq i \leq n} |i - \pi_i|.$$

Notre exemple nous donne :

$$\begin{aligned} Max(\pi^1) &= \max\{|1-7|, |2-3|, |3-2|, |4-4|, |5-9|, |6-5|, \\ &\quad |7-8|, |8-1|, |9-6|, |10-10|\} \\ &= \max\{6, 1, 1, 0, 4, 1, 1, 7, 3, 0\} \\ &= 7 \end{aligned}$$

Max est bien une mesure de désordre, car :

1. $\forall\ \pi \in S_n : Max(\pi) \geq 0$ et $Max(\pi) = 0 \Leftrightarrow \pi = \iota$ (trivial)
2. $\forall\ \pi \in S_n : Max(\pi) = Max(\pi^{-1})$: par la définition 2.2, nous avons $\forall\ 1 \leq i \leq n$

$$i - \pi_i = \pi^{-1}_{\pi_i} - \pi_i = -(\pi_i - \pi^{-1}_{\pi_i})$$

où π_i est à présent une position dans π^{-1}. Max se basant sur des valeurs absolues, nous avons bien $Max(\pi) = Max(\pi^{-1})$.

3. $\forall\ \pi, \sigma \in S_n : Max(\pi \circ \sigma) \leq Max(\pi) + Max(\sigma)$: en effet

$$\begin{aligned} Max(\pi \circ \sigma) &= \max_{1 \leq i \leq n} |i - (\pi \circ \sigma)_i| = \max_{1 \leq i \leq n} |i - \pi_{\sigma_i}| = \max_{1 \leq i \leq n} |i - \sigma_i + \sigma_i - \pi_{\sigma_i}| \\ &\leq \max_{1 \leq i \leq n} (|i - \sigma_i| + |\sigma_i - \pi_{\sigma_i}|) \\ &\leq \max_{1 \leq i \leq n} |i - \sigma_i| + \max_{1 \leq j \leq n} |\pi_j - \pi_{\sigma_j}| \end{aligned}$$

et $Max(\pi)+Max(\sigma) = \max_{1 \leq i \leq n} |i-\pi_i| + \max_{1 \leq i \leq n} |i-\sigma_i|$. Comme $\max_{1 \leq j \leq n} |\sigma_j - \pi_{\sigma_j}| = \max_{1 \leq j \leq n} |j - \pi_j|$, l'inégalité triangulaire est bien vérifiée.

6.4.3 *Exc*

Exc est le nombre minimal d'échanges de deux éléments distincts requis pour trier une permutation ; contrairement à Inv_2, les éléments échangés peuvent mais ne doivent pas être adjacents dans π. Elle est également appelée *distance de Cayley* car c'est lui qui a découvert [5] que

$$Exc(\pi, \sigma) = |\pi| - \text{ nombre de cycles dans } \pi \circ \sigma^{-1}$$

où le mot "cycles" est pris dans le sens classique décrit au chapitre 2. L'identité ci-dessus se justifie de la façon suivante : pour passer de π à σ, deux éléments π_i, π_j de π doivent être échangés si $(\pi_i, \pi_j) \neq (\sigma_i, \sigma_j)$; un échange n'a d'intérêt que s'il place au moins un des éléments sur lesquels il agit en bonne position, donc si nous nous trouvons dans l'un des trois cas suivants :

- $(\pi_i, \pi_j) = (\sigma_j, \sigma_i)$: un échange suffit donc à "ranger" π_i et π_j ;
- $(\pi_i, \pi_j) = (\sigma_j, \sigma_k)$: il faudra ensuite échanger π_i et π_{σ_k} ;
- $(\pi_i, \pi_j) = (\sigma_k, \sigma_i)$: il faudra ensuite échanger π_j et π_{σ_k}.

Ces échanges peuvent se représenter graphiquement par des arbres binaires dont les feuilles sont les éléments de π et dont chaque nœud représente un échange ; un exemple simple de cette représentation est donné en figure 6.5. Un arbre représentera donc une suite d'échanges triant tous les éléments π constituant ses feuilles selon l'ordre imposé par σ ; les éléments qui sont déjà en bonne place constitueront un arbre vide (s'il fallait échanger un élément en bonne place, ce serait avec lui-même, donc c'est inutile).

Il reste à montrer que chacun de ces arbres correspond à un cycle de $\pi \circ \sigma^{-1}$; considérons la position i dans π : si $\pi_i \neq \sigma_i$, alors il faut trouver la position k de l'élément de σ égal à π_i. Or, $\sigma_k = \pi_i \Leftrightarrow \sigma_{\pi_i}^{-1} = k$; il faut donc échanger, dans la permutation π l'élément π_i avec l'élément $\pi_{\sigma_{\pi_i}^{-1}} = (\pi \circ \sigma^{-1})_{\pi_i}$. Une fois l'échange effectué, il faut regarder si les éléments qui l'ont subi sont tous deux en bonne position. Si ce n'est pas le cas (seul l'un d'entre eux, celui qui occupe maintenant la position qu'occupait π_i), la même procédure est réitérée jusqu'à finalement retomber sur la position de départ, ce qui correspond bien à un cycle de $\pi \circ \sigma^{-1}$; le nombre d'échanges effectués pour passer de π à σ est donc donné par $|\pi| - |arbres(\pi)|$, quantité dont il faut soustraire les "arbres vides" (donc les éléments formant un cycle à eux seuls car ils occupent la même position dans π et σ), ce qui vérifie bien $Exc(\pi, \sigma) = |\pi| - |cycles(\pi \circ \sigma^{-1})|$.

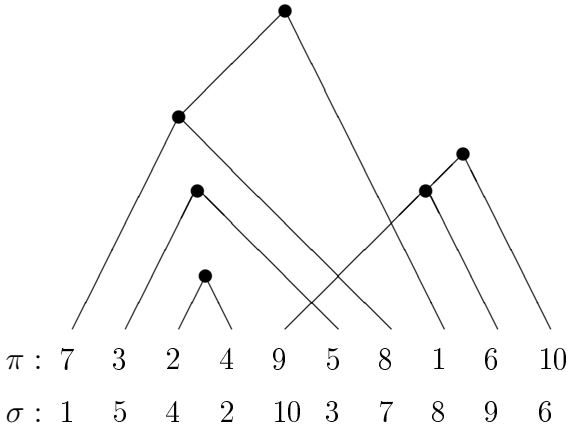


FIG. 6.5 – Les arbres binaires correspondant aux échanges à réaliser pour transformer π en σ

Le nombre d'échanges nécessaires au tri de l'exemple π^1 est donc donné par $\pi - |cycles(\pi^1 \circ \iota^{-1})|$; comme $\pi^1 \circ \iota^{-1} = \pi^1 \circ \iota = \pi^1$, dont les cycles sont (1, 7, 8), (2, 3), (4), (5, 9, 6), (10), $Exc(\pi^1) = 10 - 5 = 5$. Voici un exemple de scénario pour trier π^1 en cinq étapes (les éléments soulignés sont ceux échangés à chaque étape) :

(7324958$\underline{1}$610)
(1$\underline{32}$49587610)
(1234$\underline{95}$87610)
(12345$\underline{9}$87$\underline{6}$10)
(12345$\underline{87}$910)
(12345678910)

Exc est bien une mesure de désordre, et cela se vérifie de la même manière que pour Inv_2, en considérant cette fois-ci toutes les transpositions au lieu de seulement les transpositions d'éléments adjacents dans π.

6.4.4 *Inv*

Inv est la distance des inversions étudiée au chapitre 3, pour laquelle nous avons déjà fourni quelques exemples. Il s'agit bien d'une mesure de désordre, et cela se vérifie de la même manière que pour Inv_2 et Exc puisque en utilisant l'approche groupale (section 6.3).

A la figure 6.6 est montré le même permutoèdre, dans lequel les arêtes en pointillés relient les sommets π, σ tels que $Inv_3(\pi, \sigma) = 1$ (Inv_3 représentant le nombre d'inversions de longueur 3 nécessaires pour passer de π à σ). Le but était de construire le permutoèdre correspondant à Inv, donc de relier tout couple de sommets représentant des permutations à distance 1 l'une de l'autre (en permettant cette fois-ci des inversions de longueurs quelconques), mais le dessin devenait illisible : j'ai donc préféré m'arrêter aux inversions de longueur 3, et laisser au lecteur le soin de rajouter les arêtes correspondant aux inversions de longueur 4 (ceci est facile : il suffit de rajouter les arêtes de la forme $\{(\pi_1\pi_2\pi_3\pi_4), (\pi_4\pi_3\pi_2\pi_1)\}$).

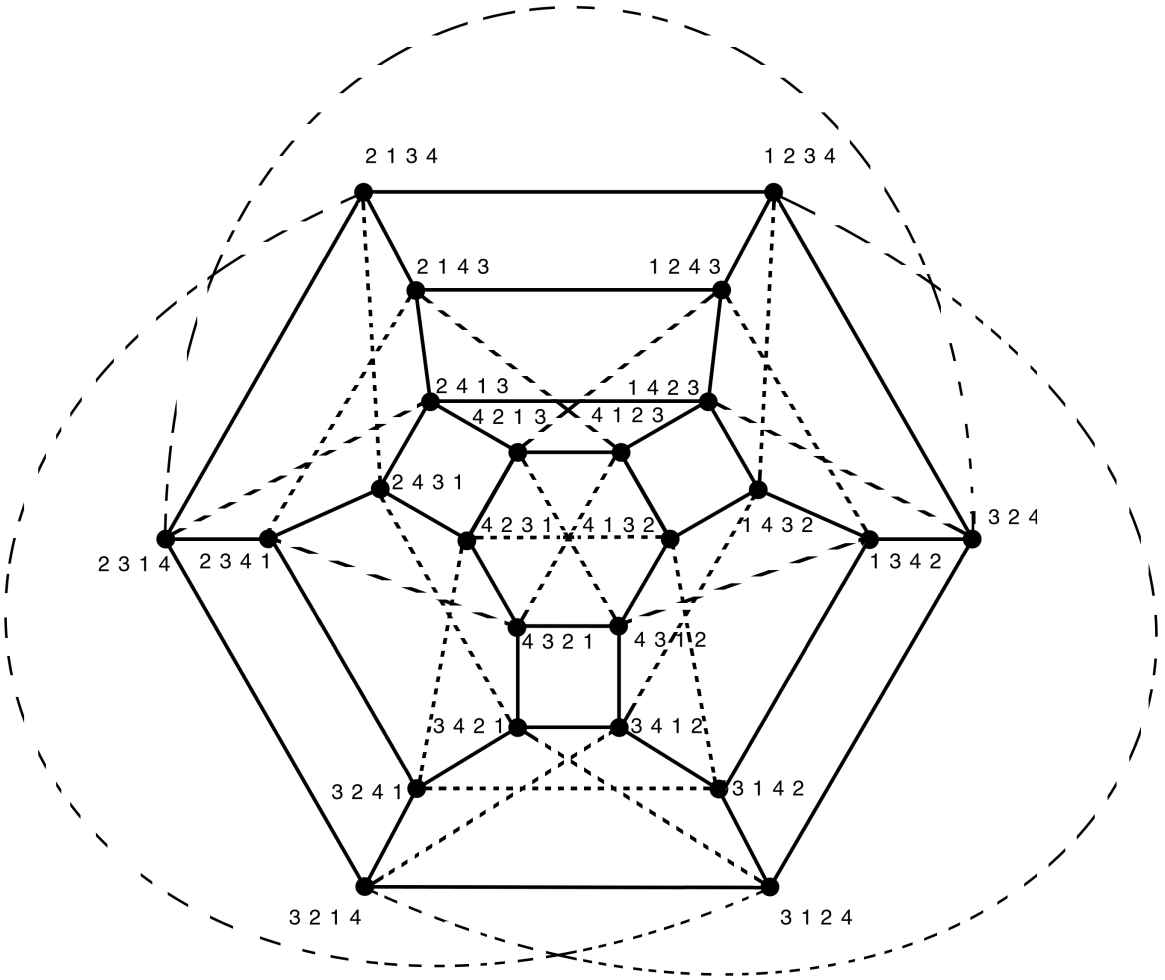


FIG. 6.6 – La même projection du permutoèdre qu'à la figure 6.3, avec en pointillés les inversions de longueur 3

Ce que nous qualifions de "mesure de désordre" dans ce chapitre est appelé par Estivill-Castro, Mannila et Wood [6] "mesure de désordre normale". Il est aussi intéressant de considérer des applications $m : S_n \to \mathbb{R}$ qui ne satisfont pas tous les axiomes de la définition 6.2. En voici quelques exemples.

6.4.5 *Dis*

Dis est la plus grande distance déterminée par un couple inversé, c'est-à-dire

$$Dis(\pi) = \max_{1 \leq i < j \leq n} \{j - i \mid (\pi_i, \pi_j) \text{ est un couple inversé}\}$$

Le plus grand écart entre deux éléments dans les couples inversés de π^1 énumérés en section 6.4.1 est de 8 positions (cf. le couple inversé (7, 6)), donc $Dis(\pi^1) = 8$.

Dis n'est pas une mesure de désordre au sens de la définition 6.2, car la symétrie n'est pas vérifiée : en effet, $Dis(\pi^1) = 8$, mais $Dis((\pi^1)^{-1}) = Dis((8\ 3\ 2\ 4\ 6\ 9\ 1\ 7\ 5\ 10)) = 9$, déterminée par le couple inversé (8, 5).

6.4.6 *Runs*

Runs est le nombre de positions auxquelles l'élément présent est inférieur à l'élément qui le précède ; représentons ces décroissances par des $\downarrow$: il suffit alors de compter le nombre de $\downarrow$ dans le vecteur de la permutation. Par exemple, $\pi^1 = (7 \downarrow 3 \downarrow 2\ 4\ 9 \downarrow 5\ 8 \downarrow 1\ 6\ 10)$, et nous obtenons $Runs(\pi^1) = 4$. Il s'agit en fait du nombre de couples inversés dont les éléments sont consécutifs dans π.

Runs n'est pas une mesure de désordre dans notre sens, car la symétrie n'est pas vérifiée : en effet, prenons $\pi = (3\ 5\ 6\ 1\ 2\ 4)$. Nous trouvons $\pi^{-1} = (4\ 5\ 1\ 6\ 2\ 3)$, et $Runs(\pi) = 1 \neq Runs(\pi^{-1}) = 2$.

6.4.7 *Osc*

Osc a été utilisée pour tenter d'améliorer, entre autres, *HeapSort* [7], et est définie comme suit :

$$Osc(\pi) = \sum_{i=1}^{|\pi|} |cross(\pi_i)|$$

où $cross(\pi_i) = \{(\pi_j, \pi_{j+1}) \mid 1 \leq j \leq |\pi| - 1 \text{ et } \pi_{j+1} < \pi_i < \pi_j\}$, c'est-à-dire l'ensemble des couples inversés d'éléments adjacents dans π et tels que π_i appartient à l'intervalle entier ouvert qu'ils délimitent. Ainsi, pour π^1 :

π_i^1	$cross(\pi_i)$
7	$(9,5),(8,1)$
3	$(8,1)$
2	$(8,1)$
4	$(7,3),(8,1)$
9	$\emptyset$
5	$(7,3),(8,1)$
8	$(9,5)$
1	$\emptyset$
6	$(7,3),(9,5),(8,1)$

Donc $Osc(\pi^1) = 12$. *Osc* n'est pas une mesure de désordre au sens de la définition 6.2, car il existe des permutations $\pi \neq \iota$ telles que $Osc(\pi) = 0$: un tel exemple est la permutation $\pi = (5\ 4\ 3\ 2\ 1)$.

6.4.8 Remarque sur les mesures de désordre

Chaque mesure de désordre sert à mesurer un type de désordre différent dans une permutation ; dans le cadre des algorithmes de tri, un algorithme est d'autant plus intéressant qu'il est optimal par rapport à plusieurs mesures de désordre différentes, et idéalement, il serait intéressant de disposer d'un algorithme optimal par rapport à toutes les mesures de désordre. Mais à l'époque de la publication de [7] (1992), un tel algorithme n'avait pas encore été découvert, et cela semble être toujours le cas car je n'ai trouvé aucun article affirmant le contraire.

Les algorithmes de tri adaptatifs s'écartant du thème de ce mémoire, nous renverrons le lecteur intéressé à [7] ; mais pour illustrer l'idée de la remarque ci-dessus,

voici quelques exemples dans lesquels certaines mesures de désordre peuvent ne pas être représentatives du désordre dans la permutation :

- si $\pi = (2\ 1\ 4\ 3\ \ldots\ i\ i+1\ \ldots\ n\ n-1)$, alors $Dis(\pi) = Max(\pi) = 1$; cela donne l'impression que π est peu désordonnée, alors que $Exc(\pi) = Inv_2(\pi) = n/2$ est bien plus représentatif du nombre d'opérations à effectuer pour trier π ;
- si $\pi = (n\ n-1\ \ldots\ 2\ 1)$, alors $Inv_2(\pi) = \binom{n}{2}$; contrairement à l'exemple précédent, π semble extrêmement désordonnée, alors que $Exc(\pi)$ montre que $n/2$ échanges suffisent à trier cette permutation ;
- comme nous l'avons déjà vu, toute permutation π dont les couples inversés d'éléments consécutifs sont également des éléments consécutifs dans ι possèdera un désordre nul selon Osc, mais cela ne signifie pas qu'elle est triée.

6.5 Adaptation de quelques distances invariantes à droite au cas signé

Le passage au cas signé va utiliser quelques notions de la théorie des groupes, que nous allons expliquer dans la section suivante. En fait, deux appoches groupales (au moins) des permutations signées sont possibles : à partir du groupe de l'hypercube et à partir d'un groupe produit direct.

6.5.1 Le groupe des isométries de l'hypercube

Pour fixer les idées, nous allons travailler avec le cube, auquel nous affectons un système de trois axes centrés au centre du cube (voir figure 6.7). Le raisonnement se généralise immédiatement à n dimensions.

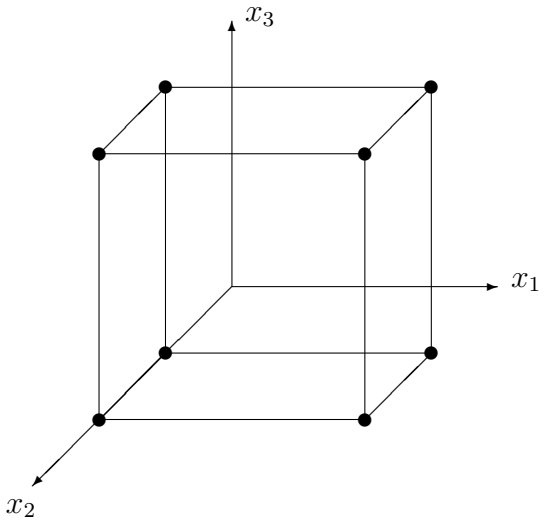


FIG. 6.7 – Un cube et les trois axes associés

Définition 6.8 *Une* isométrie *est une transformation de l'espace euclidien* $\mathbb{R}^3$ *qui conserve les distances. Elle est une* isométrie du cube *si elle conserve le cube de sommets* $(\pm 1, \pm 1, \pm 1)$.

Toute isométrie du cube est le produit d'une permutation des axes et d'une transformation particulière de la forme :

$$\begin{cases} x'_1 &= \pm\, x_1 \\ x'_2 &= \pm\, x_2 \\ x'_3 &= \pm\, x_3 \end{cases}$$

Nous décrirons cette transformation de manière plus concise en écrivant : $(\pm, \pm, \pm)$. Par exemple, nous pourrions changer l'orientation de l'axe x_2 uniquement, et cela s'écrirait $(+, -, +)$. Toute isométrie du cube peut donc s'écrire sous la forme

$$f = (\pm, \pm, \pm) \circ \alpha$$

où α est une permutation des axes.

La composition de deux isométries $f = (\pm, \pm, \pm)_f \circ \alpha$ et $g = (\pm, \pm, \pm)_g \circ \beta$ se fait de la manière suivante :

$$\begin{aligned} g \circ f &= (\pm, \pm, \pm)_g \circ \beta \circ (\pm, \pm, \pm)_f \circ \alpha \\ &= (\pm, \pm, \pm)_g \circ \underbrace{\beta \circ (\pm, \pm, \pm)_f \circ \beta^{-1}}_{\text{conjuguée, également de la forme } (\pm,\pm,\pm)} \circ \beta \circ \alpha \end{aligned}$$

Le résultat de $\beta \circ \alpha$ est connu, puisqu'il s'agit simplement de la composition de deux permutations classiques ; la composée des transformations de la forme $(\pm, \pm, \pm)$ consiste en une multiplication composante à composante et la seule difficulté réside donc dans le calcul de la conjuguée $\beta \circ (\pm, \pm, \pm)_f \circ \beta^{-1}$. Cette dernière est obtenue en *transportant* par β les flèches de $(\pm, \pm, \pm)_f$. Un exemple général d'un tel transport est donné à la figure 6.8, illustrant la conjuguée de δ par β.

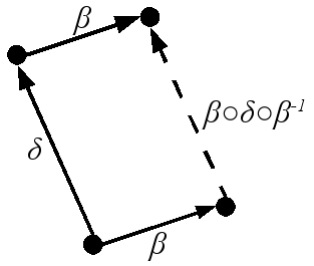


FIG. 6.8 – Le transport de la flèche δ par β

Dans notre cas, le calcul de $\beta \circ (\pm, \pm, \pm)_f \circ \beta^{-1}$ se fait en appliquant la permutation β au vecteur $(\pm, \pm, \pm)_f$.

Exemple : si $\beta = (3\ 1\ 2) = (1, 3, 2)$, $f = (-, +, +) \circ \alpha$ et $g(+, -, -) \circ \beta$, alors :

$$\begin{aligned} g \circ f &= (+, -, -) \circ \beta \circ (-, +, +) \circ \alpha \\ &= (+, -, -) \circ \beta \circ (-, +, +) \circ \beta^{-1} \circ \beta \circ \alpha \\ &= (+, -, -) \circ (+, +, -) \circ \beta \circ \alpha \\ &= (+, -, +) \circ \beta \circ \alpha \end{aligned}$$

Revenons au cas général des permutations signées de longueur n et adoptons les notations suivantes :

- π^+ et σ^+ pour les versions non signées respectives des permutations signées π et σ, c'est-à-dire les permutations obtenues à partir de π et σ en prenant leurs éléments en valeurs absolues ;
- $(\pm, \pm, \cdots, \pm)_\pi$ et $(\pm, \pm, \cdots, \pm)_\sigma$ pour les vecteurs contenant les signes des éléments des permutations signées π et σ ;
- $((\pm, \pm, \cdots, \pm)_\sigma)^{\pi^+}$ pour la conjuguée $\pi^+ \circ (\pm, \pm, \cdots, \pm)_\sigma \circ \pi^{+^{-1}}$;

Le produit des permutations signées π et σ s'exprime alors de la façon suivante :

$$\pi \circ \sigma = (\pm, \pm, \cdots, \pm)_\pi \circ ((\pm, \pm, \cdots, \pm)_\sigma)^{\pi^+} \circ (\pi^+ \circ \sigma^+) \tag{6.2}$$

La théorie groupale présentée dans la section 6.3 s'étend directement à un groupe quelconque $G, *$: on définit les mesures de désordre sur $G, *$ et les distances invariantes à droite sur $G, *$ de manière entièrement analogue. La correspondance entre ces deux concepts (exprimée dans l'équation (6.1)) se retrouve, de même que la fabrication d'une mesure de désordre à partir d'un ensemble générateur S de G satisfaisant $S^{-1} = S$.

En particulier, différentes mesures de désordre s'obtiennent de cette manière pour les permutations signées. Nous identifions pour cela l'ensemble $S_n^\pm$ des permutations signées à l'ensemble des isométries de l'hypercube de $\mathbb{R}^n$, en choisissant l'hypercube de sommets $(\pm 1, \pm 1, \cdots, \pm 1)$. Nous obtenons ainsi le groupe $S_n^\pm, \circ$ dont la loi est donnée par (6.2).

Tout ensemble S de générateurs de $S_n^\pm, \circ$ tel que $S = S^{-1}$ produit une mesure de désordre et aussi une distance invariante à droite sur $S_n^\pm, \circ$. Celle-ci est donc une distance intéressante entre permutations signées.

Voici des choix possibles : prendre S formé :

1. de la symétrie bilatérale $s_0 = (+, +, \cdots, +, -)$ et des transpositions d'éléments adjacents (au sens de la section 6.3) ;
2. de s_0 comme ci-dessus et de toutes les transpositions ;
3. de s_0 et de toutes les inversions $\rho(i, j)$;
4. de toutes les symétries bilatérales $(+, +, \cdots, +, -, +, +, \cdots, +)$ et de l'un des trois ensembles de générateurs de S_n sélectionnés ci-dessus.

Beaucoup d'autres choix sont encore possibles pour S, qui donneront encore d'autres distances entre permutations signées (toutes invariantes à droite pour la structure du groupe des isométries). Nous ne savons pas si la distance des inversions est bien invariante à droite dans ce groupe-ci ; mais ce n'est pas grave, car elle l'est dans le groupe présenté dans la section suivante.

6.5.2 Un autre groupe pour les permutations signées

Une seconde approche pour créer des distances entre permutations signées consiste à travailler dans un autre groupe. Représentons à nouveau toute permutation signée π de longueur n sous la forme

$$(\pm, \pm, \cdots, \pm) \circ \pi^+$$

(cf.section 6.5.1). Définissons un autre produit $*$ sur l'ensemble $S_n^{\pm}$ des permutations signées de longueur n en posant, pour $\pi, \sigma \in S_n^{\pm}$:

$$\pi * \sigma = (\pm, \pm, \cdots, \pm)_{\pi} \circ (\pm, \pm, \cdots, \pm)_{\sigma} \circ \pi^{+} \circ \sigma^{+}$$

Donc $(\pi \circ \sigma)^{+} = \pi^{+} \circ \sigma^{+}$ et le vecteur de signes $(\pm, \pm, \cdots, \pm)_{\pi\circ\sigma}$ s'obtient en multipliant composante à composante les vecteurs de signes $(\pm, \pm, \cdots, \pm)_{\pi}$ et $(\pm, \pm, \cdots, \pm)_{\sigma}$. Cette loi $*$ érige $S_n^{\pm}$ en un groupe. La théorie groupale de la section 6.3 s'étend à ce groupe $S_n^{\pm}, *$: le choix d'un ensemble S engendrant $S_n^{\pm}, *$ et tel que $S = S^{-1}$ permet d'introduire une mesure de désordre sur $S_n^{\pm}, *$ (dont la définition est similaire à celle donnée pour le groupe $S_n, \circ$).

Dans cette approche, la distance des inversions entre permutations signées provient de la mesure de désordre déduite d'un ensemble particulier S de générateurs de $S_n^{\pm}, *$. Il suffit en effet de prendre S formé de tous les éléments de la forme

$$(+ + \cdots + \overbrace{-}^{i} - \cdots \overbrace{-}^{j} + + \cdots +) \circ \rho(i, j)$$

où $\rho(i, j)$ désigne l'inversion comme dans la section 6.3. La distance des inversions est donc une distance invariante à droite dans le groupe $S_n^{\pm}, *$.

D'autres distances invariantes à droite s'obtiennent également en faisant un autre choix d'ensemble générateur S (avec la restriction $S^{-1} = S$). Nous laissons ceci à l'imagination du lecteur.

Chapitre 7

Conclusions

Nous avons parlé de diverses distances entre des permutations, aussi bien dans le cas signé que non signé. Nous avons évoqué diverses motivations, dont la principale est l'utilisation de la distance des inversions en bioinformatique pour la comparaison de génomes.

Cette distance est remarquable dans le sens où son calcul est NP-difficile dans le cas non-signé, mais polynomial dans le cas signé ; nous avons montré la preuve de ces deux affirmations, par la synthèse de l'article de Caprara [4] pour la première et en présentant divers algorithmes polynomiaux pour la seconde. Ceci nous a amené à décrire diverses notions combinatoires sur lesquelles reposent les algorithmes sous forme d'une synthèse de plusieurs articles.

Nous avons implémenté un des algorithmes linéaires permettant de calculer la distance des inversions dans le cas signé, et avons réalisé une étude statistique à l'aide du code ; nous avons pu en déduire de nouvelles relations entre les inversions, ainsi que des résultats des dénombrement et une mise en évidence d'une permutation positive particulière permettant d'atteindre la valeur maximale de la distance des inversions dans le cas signé. Nous avons également formulé deux conjectures :

1. il n'existe pas de permutation positive à distance 4 de l'identité ;
2. pour toute permutation signée de n éléments, où n est fixé et comprend les deux encadrements, la distance de plus grande fréquence est $n-2$ si n est impair, $n-3$ sinon.

Enfin, nous avons présenté par une synthèse de [6, 7] des distances particulières, et montré leur intérêt ; nous avons également montré comment ces distances pourraient être adaptées au cas signé. Un mathématicien pourrait sans doute formuler une théorie abstraite générale. Nous avons comparé les versions signées et non signées : il ne semble pas utopique de penser que cette approche pourrait révéler encore plus de propriétés intéressantes, comme nous l'a montré l'exemple de la distance des inversions.

Annexe A

Code source

Le programme présenté ci-dessous a été utilisé dans le chapitre 5 et génère toutes les permutations encadrées de n éléments avec leurs distances des inversions, calculées en temps linéaire par l'algorithme de [3]. Il a été exécuté dans un environnement GNU/Linux et compilé avec gcc-3.3, sans aucune option particulière.

A.1 `main.cpp`

Ce fichier contient le point d'entrée principal du programme ; l'exécutable sera lancé par `main -n` suivi de la valeur de n, un entier non signé. Ceci générera toutes les permutations encadrées de n éléments (en fait, $n-2$ éléments puisque 0 et $n-1$ sont les encadrements) avec leurs distances respectives, et stockera dans le vecteur `distances` les fréquences des différentes distances.

```
1  /*
2   * Anthony Labarre                                    2de licence Informatique
3   *                                          Université Libre de Bruxelles 2003-2004
4   *
5   * Programme générant, pour un n donné, toutes les permutations signées de n-2
6   * éléments, encadrées par 0 et n-1, calculant leur distance des inversions et
7   * stockant les fréquences de chaque valeur dans un vecteur.
8   */
9
10 #include <iostream>
11 #include <stdio.h>          // atoi()
12 #include <string.h>         // analyse des paramètres passés en lignes de commande
13
14 using namespace std;
15
16 #include "permutation.h"          // classe Permutation
17 #include "reversal_distance.h"    // classe réalisant le calcul de la distance des
18                                   // inversions d'une permutation
19
20 // Affichage du temps total mis pour générer les permutations et calculer leur
21 // distance; dans un environnement UNIX, ceci est inutile: il suffit de taper
22 // time avant la commande pour lancer l'exécutable, et le temps pendant lequel
23 // le programme aura tourné s'affichera après l'exécution.
24 //#define MEASURE_TIME_ELAPSED
25 #ifdef MEASURE_TIME_ELAPSED
26     #include <time.h>
27 #endif
28
29 // Variables globales ------------------------------------------------------------
30 int *values = NULL;                    // vecteur initialisant chaque permutation
31 unsigned int index_values = 0,                 // position dans le vecteur values
```

```
32              n = 0;                                                  // nombre d'éléments
33  long unsigned int *distances = NULL;     // contient les fréquences des distances
34  long unsigned int *hurdles = NULL;          // contient les fréquences des haies
35  long unsigned int *distances_neg = NULL;  // contient les fréquences des distan-
36                                            // ces des permutations non positives
37
38  long unsigned int *distances_pos = NULL;  // contient les fréquences des distan-
39                                            // ces des permutations non positives
40
41  // flags représentant les options
42  bool write_stats = true;
43
44  // Fonctions (déclarations)-------------------------------------------------
45
46  void bad_format(char*);    // affiche les options à l'utilisateur et sort du pro-
47                             // gramme
48  void heap_rec(int);         // génère toutes les permutations encadrées positives
49  void sign_permutation();       // génère toutes les versions signées ou non d'une
50                                 // permutation
51
52  int main(int argc, char* argv[]) {
53      if(argc>0) {
54          // examiner les paramètres
55          for(unsigned int i=0; i<argc; ++i) {
56              if(!strcmp(argv[i], "-n")) {
57                  if(i+1 < argc) {
58                      ++i;
59                      n = atoi(argv[i]);
60                  }
61                  else
62                      bad_format(argv[0]);
63              }
64          }
65      }
66      else
67          // affichage des params dispos
68          bad_format(argv[0]);
69
70  #ifdef MEASURE_TIME_ELAPSED
71      clock_t start, end;
72      start = clock();
73  #endif
74
75      // initialisation de values
76      values = new int[n];
77
78      for(unsigned int i=0; i<n; ++i)
79          values[i] = i;
80
81      if(write_stats) {
82          // initialisation de distances (d majorée par n-1)
83          distances = new long unsigned int[n];
84          for(unsigned int i=0; i<n; ++i)
85              distances[i] = 0;
86
87          distances_neg = new long unsigned int[n];
88          for(unsigned int i=0; i<n; ++i)
89              distances_neg[i] = 0;
90
91          distances_pos = new long unsigned int[n];
92          for(unsigned int i=0; i<n; ++i)
93              distances_pos[i] = 0;
94
95          hurdles = new long unsigned int[n];
96          for(unsigned int i=0; i<n; ++i)
97              hurdles[i] = 0;
98
99      }
100
101     index_values = 1;
```

```
102         heap_rec(n-2);
103
104         // affichage des distances
105         cout<<"Toutes les permutations : ";
106         for(unsigned int i=0; i<n; ++i)
107             cout<<distances[i]<<" ";
108         cout<<endl;
109
110         cout<<"Permutations negatives  : ";
111         for(unsigned int i=0; i<n; ++i)
112             cout<<distances_neg[i]<<" ";
113         cout<<endl;
114
115         cout<<"Permutations positives  : ";
116         for(unsigned int i=0; i<n; ++i)
117             cout<<distances_pos[i]<<" ";
118         cout<<endl;
119
120     /*
121         cout<<"Haies : ";
122         for(unsigned int i=0; i<n; ++i)
123             cout<<hurdles[i]<<" ";
124         cout<<endl;
125     */
126         if(hurdles)        delete[] hurdles;
127         if(distances)      delete[] distances;
128         if(distances_neg)  delete[] distances_neg;
129         if(distances_pos)  delete[] distances_pos;
130         if(values)         delete[] values;
131
132     #ifdef MEASURE_TIME_ELAPSED
133         end = clock();
134         double time_elapsed = ((double) end-start)/CLOCKS_PER_SEC;
135         cout<<endl<<"Temps ecoule : ";
136
137         cout<<time_elapsed<<" secondes"<<endl;
138     #endif
139
140         return 0;
141     }
142
143     /*
144      * Algorithme Heap récursif pour générer toutes les permutations signées enca-
145      * drées; une fois la permutation non signée fixée (condition d'arrêt), on fait
146      * appel à sign_permutation(int) pour réaliser toutes les combinaisons possibles
147      * de signes sur cette permutation
148      */
149     void heap_rec(int k) {
150         if (k == 0)
151             sign_permutation();
152         else
153             for (int c = 1; c <= k; c++) {
154                 heap_rec(k-1);
155                 swap<int>(k % 2 ? values[1] : values[c], values[k]);
156             }
157     }
158
159     /*
160      * Réalise les 2^{n-2} manières de signer values.
161      */
162     void sign_permutation() {
163         int mask = 0,   // détection des bits changeant à chaque incrémentation de c
164             upper_bound = (int) ldexp(1.0, n-2);              // 2^{n-2} permutations
165         bool shift = false;
166         for(int c=0; c<upper_bound; ++c) {
167             // permutation complète
168             Permutation T(n, values);
169             ReversalDistance RD;
170
171             // l'afficher avec sa distance
```

```
172         int d = RD.Compute(T);
173
174         // si l'utilisateur veut des stats, mettre à jour les compteurs
175         distances[d] += write_stats;
176         hurdles[RD.h] += write_stats;
177
178         if(T.IsPositive())
179             distances_pos[d] += write_stats;
180         else
181             distances_neg[d] += write_stats;
182
183         // modification de values
184         mask = c ^ c+1; // les bits qui changent sont détectés par un XOR
185         for(int i = n-2; i>0 && mask>0; --i, mask = mask >> 1) {
186             shift = mask % 2;
187             if(shift)
188                 values[i] *= -1;
189         }
190     }
191 }
192
193 /*
194  * Affiche à l'utilisateur comment utiliser le programme, puis le ferme.
195  */
196 void bad_format(char* program_name) {
197     cout<<"Utilisation: "<<program_name<<" -n naturel"<<endl;
198     exit(-1);
199 }
```

A.2 permutation.h

Ce fichier contient la déclaration de la classe `Permutation`.

```
1  /*
2   * Anthony Labarre                                    2de licence Informatique
3   *                                          Université Libre de Bruxelles 2003-2004
4   *
5   * Classe implémentant une permutation signée (déclaration).
6   */
7
8  #ifndef __PERMUTATION_H
9  #define __PERMUTATION_H
10
11 class Permutation {
12     private:
13
14 // Méthodes privées
15         void Copy(const Permutation&);                    // copie une permutation
16         void Erase();                                          // efface pi et n
17
18 // Opérateurs, classes et fonctions amis
19         friend ostream& operator<<(ostream&, Permutation&);       // affiche pi
20
21         friend Permutation reversal(Permutation, unsigned int, unsigned int);
22         friend Permutation oriented_reversal(Permutation, int, int);
23
24         friend class ReversalDistance;
25
26     public:
27 // Membres
28         int *pi;                                  // vecteur contenant la permutation
29         unsigned int n;                     // nombre d'éléments de la permutation
30
31 // Constructeurs
32         Permutation();
33         Permutation(unsigned int);                // crée la permutation identité
34         Permutation(const Permutation&);                // constructeur de copie
```

```
35          Permutation(unsigned int, int[]);
36
37  // Méthodes publiques
38          bool IsPositive();     // renvoie vrai si tous les éléments sont positifs
39
40  // Opérateurs
41          const Permutation& operator= (const Permutation&);
42          bool operator== (const Permutation&);
43          int& operator[] (unsigned int) const;  // accès direct et assignation
44                                                 // (n'utiliser cette dernière que
45                                                 // hors de la classe)
46
47          ~Permutation();
48  };
49
50  #include "permutation.cpp"
51
52  #endif
```

A.3 permutation.cpp

Ce fichier contient la définition de la classe `Permutation`.

```
1   /*
2    * Anthony Labarre                                    2de licence Informatique
3    *                                          Université Libre de Bruxelles 2003-2004
4    *
5    * Classe implémentant une permutation signée (définition).
6    */
7
8   #include "permutation.h"
9   #include <stdlib.h>      // abs()
10
11  // Méthodes privées ----------------------------------------------------------------
12  void Permutation::Copy(const Permutation& P) {
13      n = P.n;
14      pi = new int[n];
15      for(unsigned int i=0; i<n; ++i)
16          pi[i] = P.pi[i];
17  }
18
19  // Réinitialise la permutation (efface tout).
20  inline void Permutation::Erase() {
21      n = 0;
22      if(pi)
23          delete[] pi;
24  }
25
26  // Affichage de la permutation: "P_1  P_2  P_3  ...  P_n", avec deux blancs en-
27  // tre chaque élément (l'un deux sera remplacé le cas échéant par un signe -).
28  ostream& operator<< (ostream &out, Permutation &P) {
29      for(unsigned int i=0; i<P.n; ++i) {
30          // formatage spécial pour aligner les nombres positifs et négatifs
31          if(P.pi[i]<0)
32              out<<P.pi[i]<<" ";
33          else
34              out<<" "<<P.pi[i]<<" ";
35      }
36      return out;
37  }
38
39  // Méthodes publiques --------------------------------------------------------------
40
41  // Renvoie vrai si la permutation ne comporte pas d'élément négatif.
42  bool Permutation::IsPositive() {
43      bool retval = (pi[0] >=0);
44      for(unsigned int i=1; i<n && retval; ++i)
```

```
45          retval = (pi[i] >=0);
46      return retval;
47  }
48
49  // Constructeurs, destructeur ----------------------------------------------
50
51  // Crée une permutation vide.
52  Permutation::Permutation() {
53      n = 0;
54      pi = NULL;
55  }
56
57  // Crée une permutation identité de size éléments.
58  Permutation::Permutation(unsigned int size) {
59      n = size;
60      pi = new int[n];
61      for(unsigned int i=0; i<n; ++i)
62          pi[i] = i;
63  }
64
65  // Constructeur de copie.
66  Permutation::Permutation(const Permutation &P) {
67      Copy(P);
68  }
69
70  // Crée une permutation de size éléments, en y copiant array; l'utilisateur fera
71  // attention ce que array soit bien une permutation, car le constructeur ne vé-
72  // rifie pas cela!
73  Permutation::Permutation(unsigned int size, int array[]) {
74      n = size;
75      pi = new int[n];
76      for(unsigned int i=0; i<n; ++i)
77          pi[i] = array[i];
78  }
79
80  // Destructeur.
81  Permutation::~Permutation() {
82      Erase();
83  }
84
85  // Opérateurs --------------------------------------------------------------
86
87  // Assignation de la permutation avec P.
88  const Permutation& Permutation::operator= (const Permutation &P) {
89      if(this != &P) {
90          Erase();
91          Copy(P);
92      }
93      return *this;
94  }
95
96  // Renvoie vrai si deux permutations sont égales.
97  bool Permutation::operator== (const Permutation &P){
98      bool retval = (n == P.n);
99
100     for(unsigned int i=0; i<n && !retval; ++i)
101         retval = (pi[i] == P.pi[i]);
102
103     return retval;
104 }
105
106 // Accès à l'élément i grâce à P[i].
107 inline int& Permutation::operator[] (unsigned int i) const {
108     if(i>=n) {
109         cout<< "\n*** Permutation[] overflow ***\n";
110         exit(-1);
111     }
112     return pi[i];
113 }
```

A.4 pair.h

Ce fichier contient la déclaration de la classe `Pair`.

```
1  /*
2   * Anthony Labarre                                    2de licence Informatique
3   *                                          Université Libre de Bruxelles 2003-2004
4   *
5   * Classe représentant un couple (déclaration).
6   */
7
8  #ifndef __PAIR_H
9  #define __PAIR_H
10
11 class Pair {
12     private:
13         friend ostream& operator<<(ostream&, Pair&);              // affichage
14
15     public:
16         int i, j;
17
18         Pair(int, int);
19
20         void Sort();
21
22         ~Pair();
23 };
24
25 #include "pair.cpp"
26
27 #endif
```

A.5 pair.cpp

Ce fichier contient la définition de la classe `Pair`.

```
1  /*
2   * Anthony Labarre                                    2de licence Informatique
3   *                                          Université Libre de Bruxelles 2003-2004
4   *
5   * Classe représentant un couple (définition).
6   */
7
8  #include "pair.h"
9
10 // Constructeur,
11 Pair::Pair(int _i, int _j) {
12     i = _i;
13     j = _j;
14 }
15
16 // Destructeur.
17 Pair::~Pair() {}
18
19 // Trie les bornes par ordre croissant.
20 inline void Pair::Sort() {
21     if(i>j) { // swap
22         int t = i;
23         i = j;
24         j = t;
25     }
26 }
27
28 // Affiche le couple de la façon suivante: "(i, j)"
29 ostream& operator<<(ostream &out, Pair &P) {
30     out<<"("<<P.i<<", "<<P.j<<")";
31     return out;
```

```
32 }
```

A.6 reversal_distance.h

Ce fichier contient la déclaration de la classe `ReversalDistance`.

```
1  /*
2   * Anthony Labarre                                    2de licence Informatique
3   *                                        Université Libre de Bruxelles 2003-2004
4   *
5   * Classe implémentant l'algorithme de calcul de la reversal distance (déclara-
6   * tion).
7   */
8
9  #ifndef __REVERSAL_DISTANCE
10 #define __REVERSAL_DISTANCE
11
12 #include <list>
13 #include "pair.h"
14 #include "permutation.h"
15
16 class ReversalDistance {
17     private:
18 // Membres privées
19         bool *signs;                // les signes des éléments de la permutation
20         list<Pair> hurdles,                                          // haies de P
21                    unoriented_components;       // composantes non orientées de P
22
23         Permutation P;     // copie de la permutation dont on calcule la distance
24
25 // Méthodes privées
26         void FindConnectedComponents();
27         void FindCycles();
28         void FindElementaryReversals();
29         bool IsFortress();
30         unsigned int NumberOfHurdles();
31         void TransformAndStoreSigns();
32
33     public:
34         bool f;                                          // vrai si P est une forteresse
35         int c,                                           // nombre de cycles
36             h,                                           // nombre de haies
37             m,                                           // = P.n - 1
38             number_of_super_hurdles;                     // nombre de super haies
39
40         ReversalDistance();
41
42         int Compute(Permutation);
43
44         ~ReversalDistance();
45 };
46
47 #include "reversal_distance.cpp"
48
49 #endif
```

A.7 reversal_distance.cpp

Ce fichier contient l'implémentation de la classe `ReversalDistance`, qui réalise le calcul de la distance des inversions d'une permutation signée donnée selon l'algorithme linéaire de [3].

```
1  /*
2   * Anthony Labarre                                    2de licence Informatique
3   *                                          Université Libre de Bruxelles 2003-2004
4   *
5   * Classe implémentant l'algorithme de calcul de la reversal distance (défini-
6   * tion).
7   */
8
9  #include <list>
10 #include <stack>
11
12 #include "reversal_distance.h"
13
14 //#define DEBUG3
15
16 #define MINUS   0
17 #define PLUS    1
18
19 #define RIGHT   1
20 #define LEFT    0
21
22 // Méthodes privées ------------------------------------------------------------
23
24 //#define dbgfc
25
26 /*
27  * Stockage des signes de la permutation dans le vecteur signs, et transforma-
28  * tion de P en une permutation positive
29  */
30 void ReversalDistance::TransformAndStoreSigns() {
31     for(unsigned int i=0; i<P.n; ++i) {
32         if(P[i] >= 0)
33             signs[i] = PLUS;
34         else {
35             signs[i] = MINUS;
36             P[i] *= -1;
37         }
38     }
39 }
40
41 /*
42  * Détection et compte des cycles.
43  */
44 void ReversalDistance::FindCycles() {
45     Permutation positions(P.n);                          // permutation inverse de P
46     bool points[P.n];                              // les points de P (visités ou non)
47
48     // initialisation de la permutation inverse
49     for(unsigned int i=0; i<P.n; ++i) {
50         positions[P[i]] = i;
51         points[i] = true;
52     }
53 //#define dbgfc
54
55     bool origin = RIGHT;     // indique si l'on part de la gauche ou de la droite
56     unsigned int newpos = 0,
57                  save_i = 0; // élément sur lequel il faudra retomber pour clore
58                              // le cycle courant
59
60     for(unsigned int i=0; i<P.n-1; ++i) {
61         if(points[i]) {                  // si le point n'a pas encore été visité
62             points[i] = false;
63             origin = RIGHT;
64             save_i = i;
65             do {   // continuer tant qu'on ne retombe pas sur le point de départ
66                 if(signs[i] == PLUS) {               // élément de départ positif
67                     if(origin == RIGHT) {        // on cherche r_i -> trouver i+1
68                         newpos = positions[P[i] + 1];          // position de i+1
69                         if(signs[newpos] == PLUS) {
70                             points[newpos-1] = false;
```

```
71                              i = newpos - 1;
72                              origin = RIGHT;
73                          }
74                          else {
75                              points[newpos] = false;
76                              i = newpos + 1;
77                              origin = LEFT;  // car on a marqué le point à gauche
78                          }
79                      }
80                      else {                  // on cherche r_{i-1} -> trouver i-1
81                          newpos = positions[P[i] - 1];          // position de i-1
82                          if(signs[newpos] == PLUS) {
83                              points[newpos] = false;
84                              i = newpos + 1;
85                              origin = LEFT;  // car on a marqué le point à gauche
86                          }
87                          else {
88                              points[newpos-1] = false;
89                              i = newpos - 1;
90                              origin = RIGHT;
91                          }
92                      }
93                  }
94                  else {                                  // élément de départ négatif
95                      if(origin == RIGHT) {   // on cherche r_{i-1} -> trouver i-1
96                          newpos = positions[P[i] - 1];          // position de i-1
97                          if(signs[newpos] == PLUS) {
98                              points[newpos] = false;
99                              i = newpos + 1;
100                             origin = LEFT;  // car on a marqué le point à gauche
101                         }
102                         else {
103                             points[newpos-1] = false;
104                             i = newpos - 1;
105                             origin = RIGHT;
106                         }
107                     }
108                     else {
109                         newpos = positions[P[i] + 1];
110                         if(signs[newpos] == PLUS) {
111                             points[newpos-1] = false;
112                             i = newpos - 1;
113                             origin = RIGHT;
114                         }
115                         else {
116                             points[newpos] = false;
117                             i = newpos + 1;
118                             origin = LEFT;  // car on a marqué le point à gauche
119                         }
120                     }
121                 }
122             } while(i != save_i);
123             ++c; // on vient de parcourir tout un cycle, donc on augmente c de 1
124         }
125     }
126 }
127
128 /*
129  * Détection et stockage des composantes connexes.
130  */
131 void ReversalDistance::FindConnectedComponents() {
132     stack<int> M1, // sert au calcul des M_i
133                M2, // sert au calcul des m_i
134                S1, // sert à la détection des composantes connexes du type (m ... M
                       )
135                S2; // sert à la détection des composantes connexes du type (-M
                       ... -m)
136
137     int M[P.n],   // contient les M_i
138         m[P.n],   // contient les m_i
```

```
139         max[P.n], // contient les m_i
140         min[P.n],
141         s;
142
143     bool mark1[P.n], mark2[P.n];
144
145     M1.push(P.n-1);
146     M2.push(0);
147     S1.push(0);
148     S2.push(0);
149
150     M[0]     = P.n-1;
151     m[0]     = 0;
152     max[0]   = 0;
153     min[0]   = P.n-1;
154     mark1[0] = false;
155     mark2[0] = false;
156
157     for(unsigned int i=1; i<P.n; i++) {
158         // Calcul des M_i
159         if(P[i-1] > P[i]) {
160             M[i] = P[i-1];
161             M1.push(P[i-1]);
162         }
163         else {
164             while(M1.top() < P[i])
165                 M1.pop();
166             M[i] = M1.top();
167         }
168
169         // Recherche des composantes connexes du type (m ... M)
170         while(P[i] < P[s = S1.top()] || P[i] > M[s]) {
171             S1.pop();
172
173             if(max[S1.top()] < max[s])
174                 max[S1.top()] = max[s];
175             mark1[S1.top()] |= mark1[s];
176         }
177
178         // si [s, i] est un intervalle commun encadré :
179         if(signs[i]==PLUS && P[i] == max[s=S1.top()]+1 && i-s == P[i]-P[s] && i-s
                >1){
180 #ifdef DEBUG3
181             cout<<"Composante connexe : "<<s<<" "<<i<<endl;
182 #endif
183             if(mark1[s] == false)          // composante non orientée, à stocker
184                 unoriented_components.push_front(Pair(s,i));
185             mark1[S1.top()] = false;
186         }
187
188         // And now the "reverse" algorithm
189         // Compute the m_i
190         if(P[i-1] < P[i]) {
191             m[i] = P[i-1];
192             M2.push(P[i-1]);
193         }
194         else {
195             while(M2.top() > P[i])
196                 M2.pop();
197             m[i] = M2.top();
198         }
199
200         // Recherche des composantes connexes du type (M ... m)
201         while(!S2.empty() && ((P[i]>P[s=S2.top()] || P[i]<m[s]) && s>0)) {
202             S2.pop();
203             if(!S2.empty()) {
204                 if(min[S2.top()] > min[s])
205                     min[S2.top()] = min[s];
206                 mark2[S2.top()] |= mark2[s];
207             }
```

```
208         }
209
210         if(!S2.empty() && signs[i]==MINUS && P[i] == min[s=S2.top()]-1 && i-s == P[
                s]-P[i] && i-s>1){
211 #ifdef DEBUG3
212             cout<<"Composante connexe : "<<s<<" "<<i<<endl;
213 #endif
214             if(mark2[s] == false)
215                 unoriented_components.push_front(Pair(s,i));
216             mark2[S2.top()] = false;
217         }
218
219         // Mise à jour des stacks et des marquages
220         if(signs[i]==PLUS)
221             S1.push(i);
222         else
223             S2.push(i);
224
225         max[i] = P[i];
226
227         if(!S2.empty() && min[S2.top()] > P[i])
228             min[S2.top()] = P[i];
229
230         min[i] = P[i];
231
232         if(!S1.empty() && max[S1.top()] < P[i])
233             max[S1.top()] = P[i];
234
235         if(!S1.empty())
236             mark1[S1.top()] = (signs[i] == MINUS);
237
238         if(!S2.empty())
239             mark2[S2.top()] = (signs[i] == PLUS);
240     }
241
242     unoriented_components.reverse(); // car insérées dans l'ordre inverse
243 }
244
245 /*
246  * Détection et compte des haies.
247  */
248 unsigned int ReversalDistance::NumberOfHurdles() {
249     // la première composante non-orientée est une haie
250     if(!unoriented_components.empty()) {
251         Pair previous = unoriented_components.front();
252         hurdles.push_front(previous);
253
254         unsigned int L = previous.i,
255                      R = previous.j;
256
257         unoriented_components.pop_front();
258
259         bool elements_between_2_hurdles = false;
260
261         // identification des haies
262         while(!unoriented_components.empty()) {
263             // s'il s'agit de la composante maximale
264             if(unoriented_components.front().i == 0
265             && unoriented_components.front().j == m) {
266                 if(!elements_between_2_hurdles) // alors c'est une haie
267                     hurdles.push_front(unoriented_components.front());
268             }
269             // si la composante courante contient la précédente, ce n'est pas
270             // une haie
271             else if(unoriented_components.front().i <= previous.i
272                  && unoriented_components.front().j >= previous.j)
273                  /* ce n'est pas une haie*/;
274             else { // c'est une haie
275                 // pour le test de la composante maximale
276                 if(unoriented_components.front().i > R+1)
```

```
277                     elements_between_2_hurdles = true;
278                 else if(unoriented_components.front().i == R+1)
279                     R = unoriented_components.front().j;
280 
281                 // test de la super haie :
282                 // s'il y a une composante suivante
283                 if(unoriented_components.size() >= 2) {
284                     // pour accéder à la composante suivante, on va retirer la
285                     // composante courante et la remettre en tête de liste après
286                     // (elle sera temporairement stockée dans previous)
287                     previous = unoriented_components.front();
288                     unoriented_components.pop_front();
289 
290 /* A hurdle U is a super-hurdle if the span of the following unoriented component
291 contains the span of U and does not contain the span of the hurdle preceding U */
292 
293                     if(unoriented_components.front().i <= previous.i
294                     && unoriented_components.front().j >= previous.j
295                     && !(unoriented_components.front().i <= hurdles.front().i
296                     && unoriented_components.front().j >= hurdles.front().j))
297 
298                        ++number_of_super_hurdles;
299 
300                     // réinsérer notre composante
301                     unoriented_components.push_front(previous);
302                 }
303 
304                 hurdles.push_front(unoriented_components.front());
305             }
306 
307             previous = unoriented_components.front();
308             unoriented_components.pop_front();
309         }
310     }
311 
312     return (h = hurdles.size()) ;
313 }
314 
315 /*
316  * Renvoie true si la permutation est une forteresse.
317  */
318 inline bool ReversalDistance::IsFortress() {
319     return f = (h%2 && h == number_of_super_hurdles);
320 }
321 
322 // Méthodes publiques ----------------------------------------------------------
323 
324 /*
325  * Renvoie la distance de la permutation Q.
326  */
327 int ReversalDistance::Compute(Permutation Q) {
328     // initialisation des données
329     number_of_super_hurdles = c = m = h = f = 0;
330     P = Q;
331     m = P.n -1;
332     signs = new bool[P.n];
333 
334     TransformAndStoreSigns();
335     FindCycles();
336     FindConnectedComponents();
337     NumberOfHurdles();
338     IsFortress();
339 
340     if(signs) delete[] signs;
341 
342     return m /*- 1*/ - c + h + f;
343 }
344 
345 // Constructeur, destructeur ---------------------------------------------------
346 ReversalDistance::ReversalDistance() { }
```

```
347
348  ReversalDistance::~ReversalDistance() { }
```

Index

Adjacence, 10
Arête, 7
    chevauchement, 31
    parité d'une, 31
    subdivision d'une, 19
    support d'une, 31
Arrangement, 6

Chemin alterné trivial, 21
Composante connexe
    minimum et maximum, 32
    portée d'une, 32
    séparation, 32
Correspondance
    hamiltonienne, 15
    parfaite, 15
Couple
    inversé, 84
    orienté, 30
Cycle, 54
    élimination d'un, 42
    alterné, 12
      orientation d'un , 13
    amovible, 41
    chevauchement, 31
    contractile, 41
    contraction d'un, 42
    eulérien, 11
    hamiltonien, 15
    parité, 31

Décomposition en cycles alternés, 12
Distance, 7
    bi-invariante, 10
    de Cayley, 89
    des inversions, 9
    invariante à droite, 81

Forteresse, 33

Graphe, 7
    auxiliaire, 17
    bicolore, 12
    bicolore équilibré, 14
    complément d'un, 7
    complet, 7
    composante connexe d'un, 8
      orientation, 37
    connexe, 8
    de chevauchement des arêtes, 34
    de chevauchement des cycles, 31
    des cycles, 29
    des cycles (cas non signé), 11
    des cycles (cas signé), 29
      inverse d'un, 40
    des points de rupture (cas non signé), 11
    eulérien, 11
    isomorphisme, 7
    ordre d'un, 7
    simple, 7
    sous-, 8

Haie, 32
    découpe d'une, 33
    fusion, 33
    simple, 33
    super-, 33

Intervalle, 9
    commun, 47
      encadré, 49
    encadré, 38
Inversion
    élémentaire, 47, 58
      chevauchement, 48
      liaison, 54
      orientation d'une, 47
    cas non signé, 9

cas signé, 28
correspondant à un sommet impair, 36
orientée, 30
sûre, 37
score d'une, 31
symétrique d'une, 63
Isométrie, 93

MAX-ACD, 12
MAX-ECD, 11
Mesure de désordre, 81
comparaison, 82

Optimalité d'un algorithme par rapport à une mesure de désordre, 82

Partie génératrice, 82
Permutation, 6
composante connexe d'une, 49
éléments d'une, 50
orientation d'une, 50
portée d'une, 49
encadrée, 10
inverse, 6
pseudo-identitaire, 71
séquence valide, 43
maximale, 44
section d'une, 33
signée, 28
simple, 39
Permutoèdre, 85
Point de rupture, 10

Réarrangement, 6

Sommet, 7
adjacence, 7
degré d'un, 7

Tri
adaptatif, 80
par inversions
non signé, 9
signé, 28

Bibliographie

[1] Claude Berge. *Graphes et hypergraphes*. Dunod, Paris, 1970.

[2] Anne Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In *Combinatorial pattern matching (Jerusalem, 2001)*, volume 2089 of *Lecture Notes in Comput. Sci.*, pages 106–117. Springer, Berlin, 2001.

[3] Anne Bergeron, S. Heber, and Jens Stoye. Common intervals and sorting by reversals : A marriage of necessity. *Bioinformatics*, 18 :S54–S63, 2002.

[4] Alberto Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM J. Discrete Math.*, 12(1) :91–110, 1999.

[5] Persi Diaconis. *Group representations in probability and statistics*. Institute of Mathematical Statistics Lecture Notes—Monograph Series, 11. Institute of Mathematical Statistics, Hayward, CA, 1988.

[6] Vladimir Estivill-Castro, Heikki Mannila, and Derick Wood. Right invariant metrics and measures of presortedness. *Discrete Appl. Math.*, 42(1) :1–16, 1993.

[7] Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *Discrete Appl. Math.*, 1992.

[8] Cynthia Gilbas and Per Jambeck. *Developing Bioinformatics Computer Skills*. O'Reilly, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2001.

[9] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[10] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming cabbage into turnip : polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1) :1–27, 1999.

[11] B. R. Heap. Permutations by interchanges. *The Computer Journal*, 6(3) :293–298, 1963.

[12] Ian Holyer. The NP-completeness of some edge-partition problems. *SIAM J. Comput.*, 10(4) :713–717, 1981.

[13] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Comput.*, 29(3) :880–892, 2000.

[14] Haim Kaplan and Elad Verbin. Efficient data structures and a new randomized approach for sorting signed permutations by reversals. *Proceedings of the 14th Symposium on Combinatorial Pattern Matching*, 2676 :170–185, 2003.

[15] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[16] J. Meidanis, M.E.M.T. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. Technical report, Institute of Computing, University of Campinas, Campinas, Sao Paulo, Brazil, Décembre 2000.

[17] Pavel A. Pevzner. *Computational Molecular Biology.* Computational Molecular Biology. MIT Press, Cambridge, MA, 2000.

[18] Robert Sedgewick. Permutation generation methods (transparents disponibles sur `http://www.cs.princeton.edu/~rs/`).

[19] N. J. A. Sloane. Encrypting by random rotations. In *Cryptography (Burg Feuerstein, 1982)*, volume 149 of *Lecture Notes in Comput. Sci.*, pages 71–128. Springer, Berlin, 1983.

[20] N.J.A. Sloane. Online encyclopedia of integer sequences (`http://www.research.att.com/~njas/sequences/`).

[21] Eric Tannier and Marie-France Sagot. Sorting by reversals in subquadratic time. Technical report, INRIA Rhône-Alpes, Laboratoire de Biométrie et Biologie évolutive, Université Claude Bernard, Villeurbanne, France, Février 2004.