Université de Montréal

Réarrangement de génomes par inversions et analyse de l'ensemble des solutions minimales

Par

Jean-François Lefebvre

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures en vue de l'obtention du grade de Maître ès sciences (M. Sc.) en informatique

Août, 2003

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :

Réarrangements de génomes par inversions et analyse de l'ensemble des solutions minimales

Présenté par

Jean-François Lefebvre

A été évalué par un jury composé des personnes suivantes :

Miklós Csürös
Président-rapporteur

Nadia El-Mabrouk
Directrice de recherche

Anne Bergeron
Membre du jury

Mémoire accepté le 7 eptembre 2003

Résumé

Le nombre grandissant de génomes complètement séquencés permet à la génomique comparative d'avoir de plus en plus de données à analyser. Ces analyses permettent entre autres, d'approfondir notre compréhension des mécanismes d'évolution des génomes. Parmi ces mécanismes d'évolution, nous nous sommes plus particulièrement intéressé aux inversions et au calcule du nombre minimal d'inversions nécessaire à la transformation d'un génome en un autre. Hannenhalli et Pevzner ([HP95a]) ont développé un algorithme permettant de calculer cette distance d'inversion et de trouver un chemin d'inversions optimal, en temps polynomial.

Comme il existe plusieurs chemins d'inversions optimaux, nous avons étendu l'algorithme de Hannenhalli et Pevzner pour pouvoir avoir accès à tous les chemins d'inversions optimaux. Dans le cas où les deux génomes comparés ne contiennent pas les mêmes gènes, nous avons généralisé un algorithme ([EM00]) permettant de minimiser les inversions, insertions et suppressions de blocs de gènes.

Nous avons utilisé ces algorithmes pour la comparaison 2 à 2 de génomes de bactéries. Les résultats ont fait apparaître un excès de très petites inversions de un ou deux gènes. En effectuant un choix prioritaire des inversions de petites tailles à chaque étape des algorithmes, cette tendance a été accentuée. Nous avons également montré que la forte présence de ces inversions n'est pas liée à l'hypothèse de Tillier et Collins ([TC00]) concernant la symétrie des inversions par rapport à un axe de réplication.

Mots clés : réarrangement de génomes, inversion, suppression, insertion, gène singulier.

Abstract

The increasing number of totally sequenced genomes gives comparative genomics and bioinformatics large databases to analyze. These analyses can help us understand more thoroughly the mechanisms involved in genome evolution. As part of these evolutionary mechanisms, reversals and reversal distances are the main focus of this study, the reversal distance being the minimal number of reversals necessary to transform one genome into another. A breakthrough was made by Hannenhalli and Pevzner ([HP95a]) who developed a polynomial-time algorithm that calculates the reversal distance and finds an optimal reversal path.

Since there is more than one optimal reversal path, we generalized the Hannenhalli and Pevzner algorithm to be able to access all the reversal paths transforming one genome into another. Also, we generalized an algorithm developed by El-Mabrouk ([EM00]) that incorporates insertions and deletions of gene segments in the Hannenhalli and Pevzner algorithm.

We used these algorithms to compare pairs of bacterial genomes. All the results show an excess of small reversals (one or two genes). By imposing a bias towards small reversals in the algorithms, this tendency of single gene reversals has been further reinforced. We also tested if the hypothesis of Tillier and Collins ([TC00]) concerning the symmetry of reversals relatively to a replication axis, could explain our findings. Our results suggest that there is no relation between our findings and this hypothesis.

Key Words: genome rearrangement, reversal, deletion, insertion, single gene.

Table des matières

Liste des tableaux

Liste des figures

Remerciements

Je voudrais tout d'abord remercier Nadia pour le professionnalisme et le dynamisme dont elle a fait preuve tout au long de ma maîtrise. Nadia était toujours disponible pour répondre généreusement à toutes mes questions. Je lui souhaite tout le succès qu'elle mérite et la remercie encore de m'avoir permis de faire cette maîtrise.

Je tiens également à remercier Yasmine Ajana, camarade de laboratoire, dont l'expertise et l'amabilité ont été fort appréciées.

Je n'oublie pas David Sankoff et Elizabeth Tillier, que je remercie grandement pour leur collaboration à mes recherches.

Finalement, je remercie Mélanie pour son soutien moral inconditionnel et sa bonne humeur contagieuse. Je remercie aussi ma famille qui m'a toujours apporté l'aide dont j'avais besoin.

Introduction

La biologie évolutive, consistant à comprendre les mécanismes d'évolution et à établir des liens phylogénétiques entre les espèces actuelles, s'est longtemps basée sur des caractéristiques morphologiques pour avancer ses hypothèses. Cependant, ces dernières années ayant été très fructueuses au niveau génétique, de plus en plus de génomes de divers organismes sont complètement séquencés. Ainsi, l'abondance récente de nouvelles données permet d'aborder le problème de l'évolution sous un nouvel angle. La génomique comparative fait partie de cette nouvelle approche. Le traitement et l'analyse de données nécessitent le développement d'algorithmes précis et efficaces. La construction d'un arbre phylogénétique, la compréhension des mécanismes d'évolution des génomes et la reconstruction de génomes ancestraux font partie des défis de la génomique comparative et de la bioinformatique.

Le but ultime de la génomique comparative est de construire l'arbre d'évolution des espèces. Cela implique la considération de distances d'évolution particulières. La méthode traditionnelle consiste à comparer les séquences d'un même gène dans deux génomes, en considérant les mutations locales (insertion, suppression, substitution d'un nucléotide). Cependant, des gènes différents fournissent des informations différentes. Une méthode alternative consiste à comparer l'ordre des gènes dans les génomes, en considérant des mutations globales (opérations de réarrangement de blocs de gènes). Étant donné qu'une multitude de génomes sont maintenant complètement séquencés, en particulier des génomes de bactéries, il devient possible d'inférer des relations d'évolution à partir de la totalité du matériel génétique des espèces, en considérant les mutations globales. C'est le domaine des réarrangements génomiques. Depuis le début des années 90, un certain nombre de modèles de réarrangement ont été considérés. Ces modèles diffèrent surtout par le type de mutation et de données biologiques considérées. Ils donnent

lieu à des études combinatoires, algorithmiques et de théorie des graphes variées et complexes.

Nous nous intéresserons, dans ce travail, à l'analyse des distances d'inversions selon deux types d'algorithmes. Nous expliquerons tout d'abord les mécanismes de réarrangements génomiques au chapitre 1. Les génomes se distinguent en formes et en types selon l'organelle et l'espèce étudiée. Sur ces génomes, différents types de réarrangement peuvent s'effectuer, principalement des réarrangements locaux (insertion, suppression, ...) et globaux (inversion, transposition, ...). Dans le même chapitre, une revue de littérature suivra afin de voir le cheminement scientifique qui s'est effectué depuis une dizaine d'année.

Une percée dans le calcul de la distance d'inversion a été effectuée par Hannenhalli et Pevzner ([HP95a]). Ils ont publié un algorithme (algorithme HP) permettant de calculer en temps polynomial, la distance minimale d'inversion entre deux permutations signées. Leur algorithme permet de trouver une suite optimale d'inversions qui transforme un génome signé en un autre, utilisant les gènes communs aux deux organismes. Il existe cependant un grand nombre de solutions optimales et il est important de pouvoir toutes les considérer afin de mieux comprendre les mécanismes d'évolution. À cette fin, nous avons implémenté un nouvel algorithme permettant d'avoir accès à toutes les inversions possibles à chaque étape de la suite minimale d'inversion. Nous verrons donc ce nouvel algorithme, appelé « algorithme INV », dans le deuxième chapitre.

Dans le cas où les génomes comparés n'ont pas les mêmes gènes il est important de pouvoir considérer les insertions et suppressions de blocs de gènes. L'autre algorithme que nous allons voir, a été développé dans [EM00] et implémenté dans cette étude. Appelé « algorithme IIS », cet algorithme est une généralisation de l'algorithme INV permettant d'évaluer la distance d'inversion, insertion et suppression de blocs de gènes. Ce dernier ajout permet de travailler avec tous les gènes des deux génomes. Toujours dans le chapitre 2, nous expliquerons en détail

l'algorithme INV et l'algorithme IIS résultant de l'ajout des insertions et suppressions.

L'accès à l'ensemble des suites minimales d'inversion, permet d'étudier différentes hypothèses sur le mécanisme des inversions, en fixant différents critères dans la sélection des inversions effectuées. La taille des inversions constitue un critère intéressant. Sankoff ([SA02]) a mentionné l'existence d'un lien entre la taille des inversions et la conservation de groupes de gènes entre les génomes. Nous nous sommes plus particulièrement intéressé à la grande proportion des inversions de gènes singuliers. Tillier et Collins ([TC00]) ont avancé l'hypothèse que les inversions se positionnaient en fonction de l'axe de réplication chez certaines espèces de bactéries. Nous avons également vérifié si cette particularité ressortait de nos données. Tous ces résultats seront présentés dans l'étude du mécanisme des inversions au chapitre 3.

La présente recherche a donné lieu à la publication de deux articles scientifiques ([ALT02] et [LET03]). La participation de l'auteur à ([ALT02]) a consisté à développer un algorithme générant des permutations aléatoires d'une distance d'inversion donnée. Aussi, le développement de l'algorithme INV, permettant de trouver toutes les solutions sûres à la résolution des bonnes composantes, a été effectué avec le concours de Yasmine Ajana. Pour ce qui est du deuxième article, les ajustements apportés à l'algorithme INV et tous les résultats expérimentaux obtenus ont été effectués par l'auteur. De plus, le développement de l'algorithme IIS, d'inversions, insertion et suppression, inspiré de [EM00], a également été implémenté par l'auteur.

Chapitre 1 :

Réarrangements génomiques et revue de littérature

Une petite introduction à la biologie génétique est nécessaire afin de comprendre les réarrangements génomiques. Mentionnons tout d'abord que l'ADN (Acide désoxyribonucléique) est à la base de tout organisme vivant. L'ADN est composé des quatre nucléotides : Adénosine (A), Cytosine (C), Guanine (G) et Thymine (T), et est disposé sur des chromosomes. Un **chromosome** correspond à une suite de nucléotides disposés en une double hélice. Les deux hélices sont jointes ensemble par des liaisons hydrogènes entre les nucléotides A, T et les nucléotides G, C. Ces deux hélices sont dites antiparallèles puisqu'elles ont des directions opposées.

Dans un organisme vivant, l'ADN est transcrit en ARN (Acide ribonucléique) et l'ARN est ensuite traduit en protéine. Ce n'est cependant pas tout l'ADN d'un chromosome qui est transcrit. Un chromosome possède plusieurs zones fonctionnelles séparées, appelées **gènes**, et ce sont les gènes qui codent pour les différentes protéines. Voyons maintenant quels mécanismes régissent l'évolution des génomes.

1.1 Mécanismes biologiques

Le **génome** correspond à l'ensemble des chromosomes d'un organisme. Généralement, les génomes **multi-chromosomaux** sont constitués de chromosomes linéaires et les génomes **uni-chromosomaux** sont constitués d'un chromosome circulaire. Un organisme vivant peut contenir jusqu'à trois génomes. Le génome nucléaire est le plus important et le plus long, le génome mitochondrial et le génome chloroplastique sont beaucoup plus petits et ne se retrouvent que dans un type

d'organelle (mitochondrie et chloroplaste, respectivement). Lorsque l'on parle du génome d'un organisme, c'est le génome nucléaire qui est visé. Chacun de ces génomes possède une autonomie génétique faisant en sorte que leur évolution génétique est indépendante des autres. Cette indépendance fait en sorte que chacun possède ses caractéristiques génomiques propres. Ainsi, les génomes mitochondrial, chloroplastique et de bactéries, sont généralement des génomes uni-chromosomaux qui sont connus pour avoir évolué par inversions.

Dans les études des réarrangements génomiques, le génome est représenté comme une suite ordonnée de gènes symbolisés par un numéro. Généralement, un signe est attribué à chaque gène. Ce signe représente l'orientation de transcription du gène. En d'autres termes, un génome est symbolisé par une permutation signée. Dans certains cas, la méthode de décodage ne permet pas de connaître l'orientation de transcription des gènes. Dans ce cas, le génome est représenté par une permutation non signée. Un chromosome circulaire est représenté de la même façon qu'un chromosome linéaire, avec la différence que les gènes aux deux extrémités de la suite sont contigus. Par exemple, le génome circulaire [+1 +2 +3 +4] peut également être représenté par [+4 +1 +2 +3] ou par [+2 +3 +4 +1]. De plus, autant dans le cas circulaire que linéaire, il est possible d'inverser l'ordre de toute la suite à condition de ne pas oublier de changer les signes des gènes. Ainsi, la suite [+1 +2 +3 +4] peut être représentée par [-4 -3 -2 -1].

Deux génomes sont identiques lorsqu'il existe une représentation d'un génome où tous les gènes composant sa séquence sont dans le même ordre et avec le même signe que ceux composant la séquence de l'autre génome. La figure 1 montre un exemple de deux génomes différents. Dans le deuxième cas, seuls les gènes +1 et +8 sont au même endroit dans les deux suites génomiques.

Génome A	+1	+2	+3	+4	+5	+6	+7	+8
Génome B	+1	+3	+4	-5	+6	-7	+2	+8

Figure 1 : Exemple de deux séquences génomiques différentes.

Les opérations de réarrangement que nous considérons dans ce travail sont les inversions, les insertions et les suppressions. Une brève introduction aux autres mécanismes de réarrangement sera également présentée.

1.1.1 Les inversions

Les **inversions** consistent à prendre un segment et à l'inverser sans changer sa position dans le génome. Afin de visualiser comment cela peut se produire, il suffit d'imaginer que le chromosome se tord et fait une boucle sur lui-même. Ensuite, à l'intersection de la boucle, la séquence est brisée et est reformée avec le mauvais brin. Ainsi, la séquence linéaire de départ est changée.

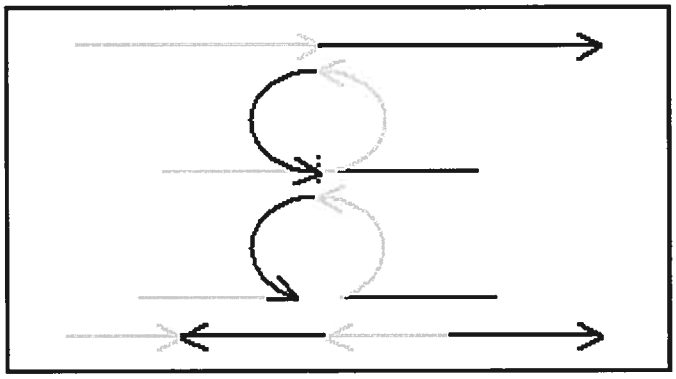

Figure 2 : Mécanisme biologique donnant lieu à une inversion.

Du point de vue de la séquence génomique, le résultat d'une inversion est qu'un segment de gènes contigus se retrouve au même endroit mais dans l'ordre inverse et avec des signes inversés (figure 3).

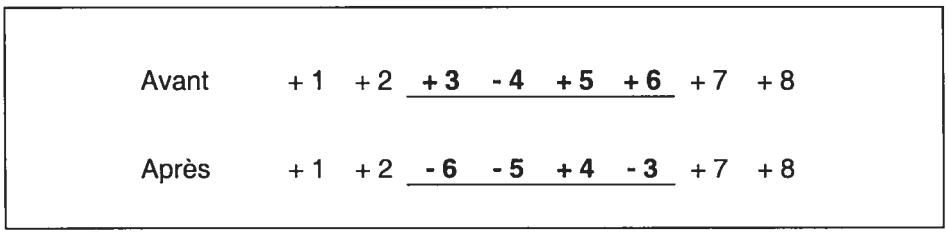

Figure 3 : Une séquence génomique avant et après qu'une inversion ait été effectuée entre les gènes +2 et +7. L'ordre et le signe des gènes sont inversés.

1.1.2 Insertions / suppressions

Une **suppression** correspond à supprimer totalement un segment de gènes d'un génome. Le mécanisme sous-jacent, tout comme dans le cas d'une inversion, consiste en la formation d'une boucle dans le chromosome. Il y a aussi coupure dans la séquence génomique mais lorsqu'elle est reformée, les brins se rassemblent de façon à laisser le contenu de la boucle hors de la séquence. Ce segment n'est alors plus compris dans la séquence génomique. Pour ce qui est de l'**insertion**, c'est le mécanisme inverse, où un segment externe à la séquence génomique, s'y insère. Ce mécanisme peut être dû à des transferts horizontaux entre différentes espèces ou à des attaques virales.

Il est cependant à noter que les suppressions sont plus fréquentes au cours de l'évolution que les insertions. Lorsqu'un gène n'est présent que dans un seul des deux génomes comparés, c'est le plus souvent dû à une suppression. C'est donc dire que les génomes ancestraux possédaient la grande majorité des gènes présents dans l'un et/ou l'autre des deux génomes. Les suppressions ont ensuite agit différemment sur les descendants, ce qui explique que certains gènes sont restés présents dans un génome et non dans l'autre.

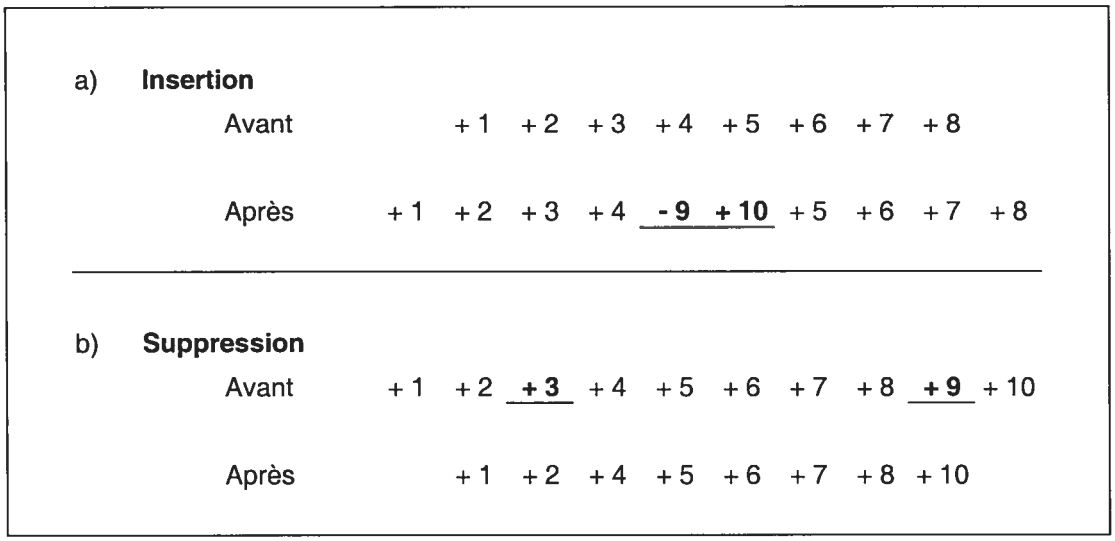

Figure 4 : Exemple d'une insertion (a) et de deux suppressions (b) sur une séquence génomique.

1.1.3 Autres mécanismes

Comme nous l'avons mentionné plus haut, il existe d'autres opérations mutagènes agissant sur les séquences génomiques, telles que les transpositions, transversions, duplications, translocations réciproques, fusions et fissions. Comme nous n'avons pas incorporé ces opérations dans notre algorithme, nous n'allons les expliquer que brièvement.

Les **transpositions** consistent à prendre un segment dans la séquence et à le placer tel quel à un autre endroit de la séquence. Le segment transposé garde donc le même sens de lecture avant et après la transposition. Ceci correspondrait à faire trois opérations d'inversion pour le même résultat.

Les **transversions** sont semblables aux transpositions à la différence qu'au lieu de se réinsérer dans le même sens, le segment est inversé. Ceci correspondrait à faire deux inversions.

Une **duplication** correspond à la création d'une copie d'un segment de gènes à l'intérieur d'un génome. Le duplicat peut être retrouvé sur le même chromosome

ou sur un autre. Un problème apporté par cette mutation est celui de retrouver le gène initial parmi toutes les copies. La position de ce gène initial dans la suite de gènes, est importante pour trouver la bonne distance d'évolution. La figure 5 montre un exemple de chacun de ces trois réarrangements.

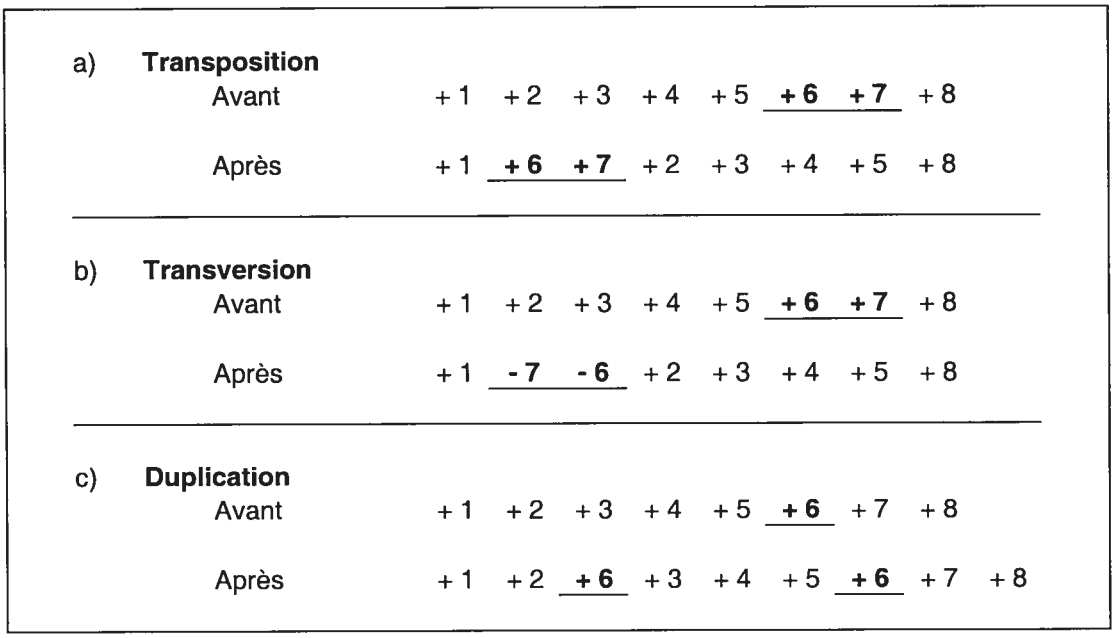

Figure 5 : Exemple d'une transposition (a), d'une translocation (b) et d'une duplication (c).

Pour ce qui est des **translocations réciproques**, **fusions** et **fissions**, ce sont des opérations qui échangent le matériel génétique entre différents chromosomes d'un même génome. Une **translocation réciproque** échange deux fragments terminaux entre deux chromosomes. Ces fragments n'ont pas nécessairement la même taille. Les **fusions** correspondent à joindre deux chromosomes en un, alors que les **fissions** correspondent à séparer un chromosome en deux.

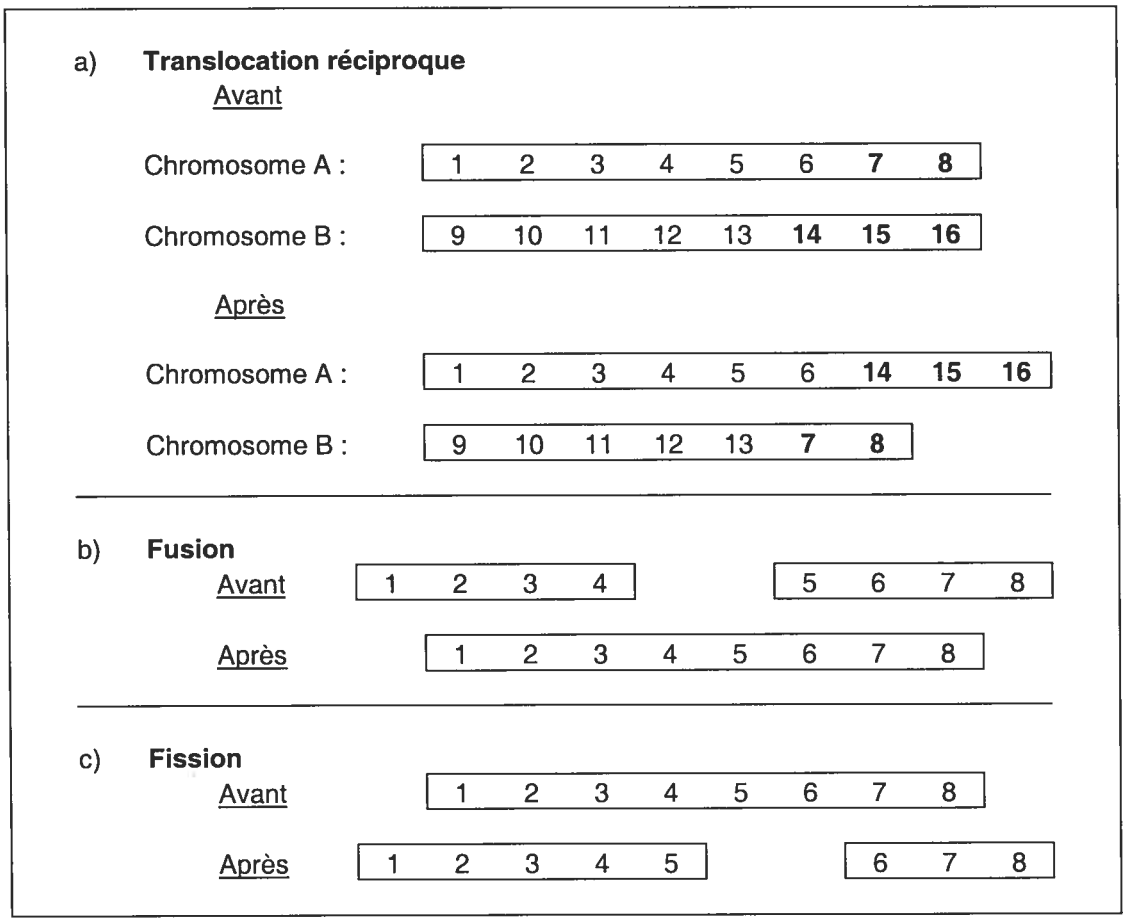

Figure 6 : Exemple d'une translocation réciproque (a), d'une fusion (b) et d'une fission (c).

1.2 Revue de littérature

Depuis une dizaine d'années, plusieurs auteurs se sont intéressés à calculer la distance d'évolution séparant deux espèces, c'est-à-dire le nombre minimal de réarrangements nécessaires pour passer d'un génome à un autre, et à expliciter des scénarios d'évolution. Parmi toutes ces opérations de réarrangement, les inversions ont été les plus considérées. La raison principale est qu'elles sont les mutations globales les plus courantes. En effet, Palmer et Herbon [PH88] ont remarqué que les

mitochondries de plusieurs espèces de plantes évoluent presque exclusivement par inversions.

- **Algorithme de Hannenhalli et Pevzner (HP)**

Trouver la distance d'inversion pour des permutations non signées de gènes a été conjecturé NP-complet par Kececioglu et Sankoff ([KS95]) dès 1995 mais, la preuve n'est venue que quatre années plus tard avec Caprara ([CA99b]). Entre-temps, Hannenhalli et Pevzner ([HP95(a)]) ont développé un algorithme calculant la distance d'inversion de gènes signés en temps polynomial. Nous verrons en détail cet algorithme dans le chapitre suivant.

L'algorithme HP a inspiré plusieurs auteurs qui se sont penchés sur son amélioration au niveau de la performance et de la précision. Caprara ([CA99a] et [CA99b]) a étudié l'algorithme afin d'en définir les bornes inférieures de temps de calcul, les approximations de temps d'exécution et les pires cas envisageables. D'autres se sont penchés sur l'amélioration de la performance en réduisant le temps de recherche des inversions sûres (définition à suivre) ([BE01]) ou le temps de recherche des solutions minimales ([BCH02], [BMY01], [CR02], [CH98], [CLN01] et [KST97]).

- **Autres distances d'évolution**

D'autres réarrangements ont été pris en compte dans la comparaison de génomes. En particulier, l'algorithme HP a été généralisé au calcul de la distance de translocation ([HA96]) et à la distance d'inversion / translocation ([HP95b]). Pour ce qui est de la distance de transposition, elle est définitivement plus difficile à étudier, et sa complexité théorique n'a pas encore été établie. Quelques auteurs ont étudié certains aspects de la distance de transposition ([BP98], [CI01] et [HA03]). Kunisawa ([KU01]) parle plutôt d'une certaine classe de transpositions appelées « transpositions déterminantes », ce qui permet de diminuer considérablement le nombre de calculs pour évaluer les distances évolutives. Se rapprochant des transpositions, Heath et Vergara ([HV00] et [HV98]) ont travaillé sur le nombre

minimal de déplacements de blocs de gènes contigus pour passer d'un génome à un autre.

Les algorithmes que nous venons de mentionner, s'appliquent à des données génétiques limitées, c'est-à-dire, que ces algorithmes ne considèrent que les gènes communs aux espèces à l'étude. El-Mabrouk ([EM00]) a élaboré un algorithme ajoutant les insertions et suppressions de segment de gènes dans l'algorithme HP, ce qui permet de comparer deux génomes ne contenant pas nécessairement les mêmes gènes. Ce dernier algorithme a été utilisé lors de la présente recherche et nous le décrirons en détail dans le prochain chapitre.

D'autres auteurs ont développé des méthodes permettant de tenir compte des familles de gènes (gènes en copies multiples) pour évaluer la distance entre deux génomes, et plus généralement pour reconstruire des arbres de phylogénie ([EM02], [SA99], [SE00] et [YCD00]).

- **Approches paramétriques**

Une autre approche à ce problème a été considérée : l'approche paramétrique. Blanchette *et al.* ([BKS96]) ont développé un algorithme (Derange II) permettant de mettre différentes fréquences sur trois opérations de réarrangement (transposition, transversion et inversion). Plus tard, Wang et Warnow ([WW01]) ont développé un algorithme probabiliste permettant de changer les probabilités d'occurrence de ces mêmes trois opérations de réarrangement et d'obtenir ce qu'ils appellent une « true evolutionnary distance » (distance évolutive réelle).

- **Comparaison d'un ensemble de génomes**

Mis à part les réarrangements, il existe d'autres caractéristiques génomiques permettant de définir des distances évolutives. La préservation de séquences de gènes à travers différents génomes, aussi appelé invariants phylogénétiques, a été utilisée par Sankoff et Blanchette ([SB99]). Le principe biologique derrière cette approche est que plus les espèces sont proches, plus leurs invariants phylogénétiques sont nombreux et longs. L'intérêt est de pouvoir travailler plus facilement sur plusieurs

génomes à la fois. Wu et Gu ([WG02]) et Bourque et Pevzner ([BP02]) ont également développé des heuristiques afin de pouvoir travailler sur plusieurs génomes à la fois.

- **Distance de points de cassure**

La distance d'inversion, de même que la distance de transposition, reste difficile à calculer pour plusieurs génomes à la fois. La construction d'arbres phylogénétiques, nécessitant le calcul des distances entre plusieurs paires d'espèces, devient donc ardue. Pour pallier à ce problème, une notion plus simple à étudier que les distances de réarrangement a été introduite : la notion de points de cassure ([WEH82]). Un point de cassure correspond à un endroit dans une permutation où deux gènes signés se suivent et où dans l'autre permutation, soit ces deux gènes ne se suivent pas ou ils n'ont pas les mêmes signes que dans la première. La figure 7 montre deux exemples pour chacune de ces situations.

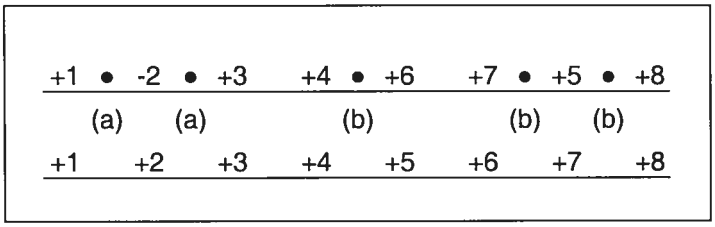

Figure 7 : Exemple de la permutation d'un génome ayant cinq points de cassure avec la permutation identité. Les points de cassure (a) sont causés par le changement de signe du gène « 2 » et les (b) proviennent du déplacement du gène « 5 ».

De ces points de cassure, une distance de points de cassure peut être calculée et utilisée à la place des distances de réarrangement. La généralisation du calcul simultané des distances séparant un génome médian à plusieurs génomes réels, est un problème NP-complet ([PS98]) mais, l'utilisation d'heuristiques a permis de le simplifier grandement ([BKS99], [BP96], [GN02], [MBW02], [SB97], [SBD00] et [SE00]).

- **Mécanisme des réarrangements génomiques**

Comme nous venons de le voir, il existe un grand nombre de méthodes permettant de reconstruire des scénarios d'évolution. Les méthodes diffèrent selon le type de données biologiques utilisées et les opérations de réarrangement considérées. L'analyse des résultats obtenus grâce à ces différents outils permet d'en déduire des caractéristiques générales des mécanismes d'évolution. Les questions auxquelles on peut tenter de répondre sont : y a-t-il des sites préférentiels pour les réarrangements? Les petites inversions sont-elles plus fréquentes que les grandes? Certaines mutations sont-elles plus fréquentes que d'autres? Nous citons ici quelques résultats obtenus.

Sankoff ([SA02]) a observé un lien pouvant exister entre la taille des inversions et la préservation de conglomérats de gènes dans les génomes. Il a remarqué que l'évolution génomique différait selon que le rapport de la taille des inversions sur la taille du conglomérat était petit ou grand. Aussi, Dalevi *et al.* ([DEE02]) ont étudié la contribution de différentes opérations de réarrangement dans l'évolution de deux espèces de bactéries *Chlamydia*. Ils ont observé que la moitié des opérations étaient des transpositions et qu'environ 40% des autres étaient des petites inversions. La comparaison de paires de bactéries a permis à Tillier et Collins ([TC00]) d'énoncer l'hypothèse que, chez les bactéries, les inversions s'effectuent préférentiellement autour d'un axe : l'axe de réplication du génome.

Tout récemment, la comparaison des génomes complets de l'homme et de la souris, a permis à Pevzner et Tesler ([PT03]) de constater que les inversions ne s'effectuaient pas de façon aléatoire (Modèle Nadeau-Taylor), mais qu'il y avait effectivement des sites préférentiels de cassure.

Chapitre 2 :

Description des algorithmes

Dans ce chapitre, nous expliquerons en détails l'algorithme HP, l'algorithme INV et l'algorithme IIS. En ce qui a trait au calcul de la distance d'inversion et à la décomposition du graphe, nous nous sommes grandement inspiré de l'algorithme HP, présenté dans l'article original de Hannenhalli et Pevzner ([HP95a]), et présenté ici en première partie. Pour ce qui est de trouver et parcourir les chemins d'inversions, nous avons élaboré l'algorithme INV, que nous présenterons en deuxième partie de ce chapitre. Quant à l'algorithme IIS décrit dans [EM00], correspondant à l'extension de l'algorithme INV pour inclure les insertions et suppressions, nous le présenterons en troisième partie.

2.1 Algorithme HP

L'algorithme HP permet de comparer deux permutations G et H contenant les mêmes gènes et de calculer la distance d'inversion correspondante (nombre minimum d'inversions nécessaires pour transformer G en H). Il permet également de trouver un chemin optimal d'inversions. Par convention, on considère le génome H comme étant l'identité (+1 +2 …+N). L'algorithme HP est basé sur un graphe appelé graphe de points de cassure. Le graphe a été introduit pour la première fois par Bafna et Pevzner ([BP96]). Nous décrivons ce graphe dans la prochaine section.

2.1.1 Graphe des points de cassure

Pour chaque gène x, on attribue deux valeurs, $2x-1$ et $2x$. Par exemple, pour le gène 5, on obtient 9 et 10. Ensuite, selon le signe du gène x, on place ses valeurs en

ordre croissant ou décroissant. Par exemple, toujours pour 5, si l'on a –5, on obtient 10 9, et si l'on a +5, on obtient 9 10. La figure 8(b) montre un exemple de cette première transformation sur un génome.

Étant donné que nous travaillons sur un génome circulaire, le dernier gène est considéré adjacent au premier. Par convention, le gène *g* dénoté par le plus grand entier, constitue l'extrémité de notre permutation (2g à une extrémité et 2g-1 à l'autre). Cette deuxième transformation, illustrée à la figure 8(c), ne change en rien la permutation.

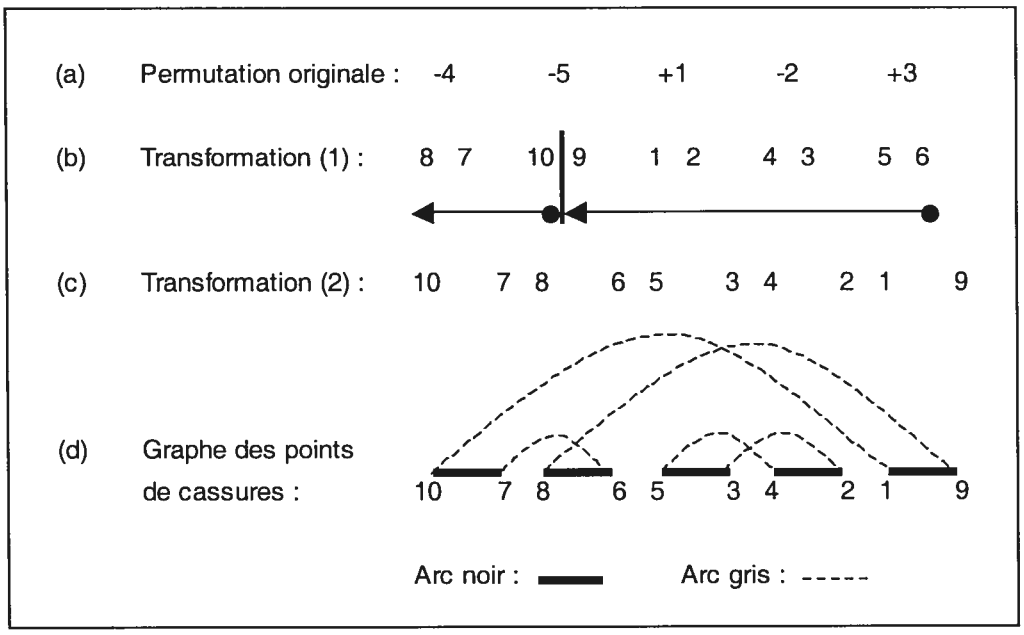

Figure 8 : Exemple de la construction d'un graphe de points de cassure. En (a), c'est la permutation signée originale du génome G. En (b), la première transformation où un gène (x) devient (2x-1 2x) ou (2x 2x-1) selon le signe de (x). En (c), on change le sens de la séquence et on sépare le dernier gène en ses deux composantes. Finalement, en (c), on construit le graphe des points de cassure avec les arcs noirs et gris.

Pour la transformation du génome G en le génome H, les arcs formant le graphe de points de cassure, sont de deux types : les **arcs noirs** qui relient les gènes adjacents dans G et les **arcs gris** qui relient les gènes adjacents dans H.

ARC NOIR : Soient les gènes *x*, *y* et *z* adjacents dans G, tels que *x* précède *y* et *y* précède *z*. Dans le graphe des points de cassure, on aura donc un arc noir reliant *2x* à *2y-1* et un arc noir reliant *2y* à *2z-1*.

ARC GRIS : Soient les gènes *x*, *y* et *z* adjacents dans G, tels que *y* précède *x* et *x* précède *z*. Dans le graphe des points de cassure, on aura donc un arc gris reliant *2y* à *2x-1* et un arc noir reliant *2x* à *2z-1*.

Dans les deux cas, aucun arc ne relie *2x* et *2x-1*, pour le même *x*. Le graphe des points de cassure ainsi obtenu est montré à la figure 8(d).

2.1.2 Décomposition du graphe en cycles et composantes

Le graphe de points de cassure se décompose naturellement en un ensemble de cycles fermés, chaque cycle étant une succession d'arcs noirs et gris alternés. Nous appelons taille d'un cycle, le nombre d'arcs noirs qu'il contient. Dans l'exemple de la figure 8(d), nous avons deux cycles : un de taille 3, contenant les arcs noirs (10-7, 8-6 et 1-9), et un de taille 2 contenant les arcs noirs (5-3 et 4-2).

On peut fixer un sens de parcours d'un cycle, et ainsi fixer une **orientation** pour les arcs du cycle. Pour ce faire, on suit le chemin formé par les arcs gris et les arcs noirs du graphe. On prend au hasard un arc noir d'un cycle et on décide du sens de départ du parcours, c'est-à-dire vers la droite ou vers la gauche. En suivant ensuite les arcs gris, on arrive à établir un sens de parcours pour chacun des arcs noirs du graphe. Ce sens de parcours est représenté par des flèches dans les figures 9 et 10. Étant donné que les arcs noirs et gris sont définis par les positions relatives des gènes de G par rapport à ceux de H, il est impossible de savoir d'avance le sens qu'aura un arc noir simplement à partir des nœuds qui le forment. Il est malheureusement nécessaire de parcourir chaque cycle pour obtenir le sens des arcs noirs.

ARCS NOIRS CONVERGENTS : deux arcs noirs sont dits **convergents** lorsque leur sens de parcours sont similaires, c'est-à-dire, les deux vers la gauche ou les deux vers la droite.

ARCS NOIRS DIVERGENTS : deux arcs noirs sont dits **divergents** lorsque leur sens de parcours sont opposés, c'est-à-dire, un vers la gauche et l'autre vers la droite.

Ainsi, pour le cycle de la figure 9, l'arc noir (a,c) diverge de (b,d). L'orientation d'un arc gris dépend de celle des arcs noirs qu'il relie. Un arc gris est dit **orienté** s'il joint deux arcs noirs divergents et il est dit **non-orienté** s'il en joint deux convergents.

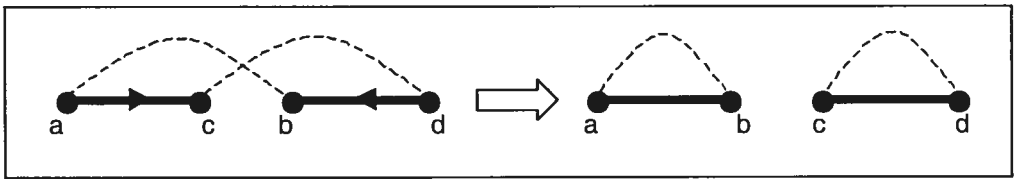

Figure 9 : Exemple d'une bonne inversion effectuée sur deux arcs divergents et créant deux nouveaux cycles à partir d'un seul.

Comme une inversion s'effectue sur deux arcs noirs, il est possible de définir le type d'une inversion en fonction de l'orientation des arcs noirs impliqués. Lorsqu'une inversion est effectuée sur deux arcs noirs divergents d'un même cycle, elle provoque la séparation de ce cycle en deux nouveaux cycles. On appelle une telle inversion une **bonne inversion** (figure 9). Si les deux arcs du même cycle convergent, l'inversion ne provoque aucun changement quant au nombre de cycles du graphe. Enfin, si une inversion est effectuée sur deux arcs noirs de deux cycles différents, ces deux cycles fusionnent alors en un seul, et l'inversion est une **mauvaise inversion**.

Le nombre maximal de cycles que peut contenir un graphe de points de cassure est égal à son nombre d'arcs noirs. Dans ce cas extrême, le graphe ne contient que des cycles de taille 1. Un graphe ne contenant que des cycles de taille 1 signifie que les deux permutations comparées sont identiques. Le problème de minimiser le nombre d'inversions se ramène donc à augmenter le plus vite possible le nombre de cycles du graphe, en effectuant le plus possible de bonnes inversions.

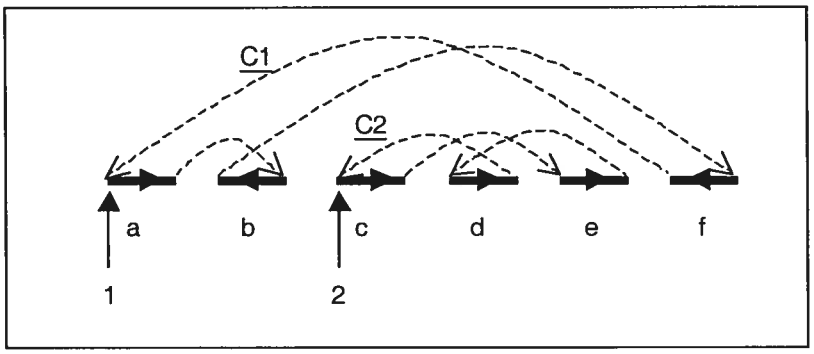

Figure 10 : Graphe de points de cassure d'une permutation contenant un cycle orienté (C1) et un cycle non-orienté (C2).

La distance d'inversion est déterminée par la décomposition du graphe en composantes telles que définies ci-dessous. On appelle **cycle orienté**, un cycle possédant au moins un arc gris orienté. Un **cycle non-orienté** est un cycle ne contenant que des arcs gris non-orientés. Dans la figure 10, le cycle C1 est orienté et le cycle C2 est non-orienté. En plus de leur orientation, les cycles peuvent se chevaucher ou non. Deux cycles se chevauchent quand au moins un arc gris de l'un croise au moins un arc gris de l'autre.

On appellera **composante**, un ensemble maximal de cycles qui se chevauchent. Une **composante orientée** (**bonne composante**) contient au moins un cycle orienté, alors qu'une **composante non-orientée** (**mauvaise composante**) n'en possède pas.

On dit qu'une composante B sépare deux composantes A et C, si tout arc virtuel qui relierait une arête de A à une arête de C, croisait un arc de B. Dans la figure 10, la composante B sépare les composantes A et C, ainsi que les composantes C et D.

Les mauvaises composantes se divisent en plusieurs catégories. Si une composante sépare d'autres mauvaises composantes, c'est un **non-obstacle**, sinon c'est un **obstacle** (hurdle). Finalement, un **super-obstacle** est un obstacle qui « protège » un non-obstacle de B, dans le sens où si l'on supprime C, B devient un obstacle. C'est le cas de C et de B de la figure 11.

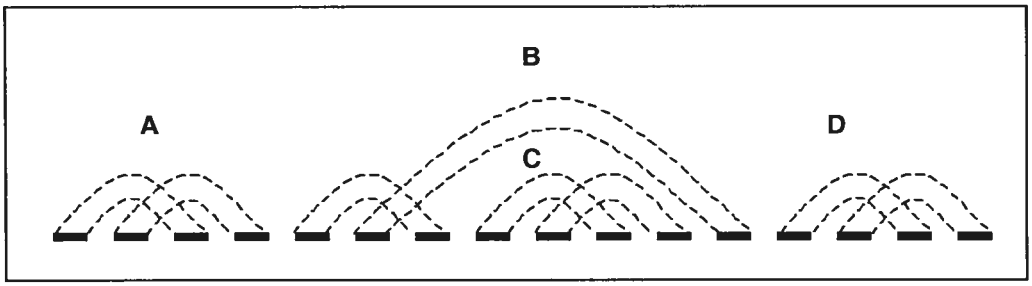

Figure 11 : Exemple de deux obstacles simples (A et D), d'un non-obstacle (B) et d'un super-obstacle (C).

Une bonne composante peut être **résolue**, c'est-à-dire transformée en cycles de taille 1, en ne faisant que de bonnes inversions, de même qu'un non-obstacle. Par contre, un obstacle nécessite une mauvaise inversion, suivie de bonnes, pour être résolu. Il existe aussi un cas particulier de graphe appelé **forteresse**. Une forteresse est un graphe contenant un nombre impair d'obstacles, tous super-obstacles. Ce cas particulier nécessite une inversion de plus. Ainsi, l'équation du nombre minimal d'inversions I(G,H) nécessaires pour passer du génome G au génome H, est la suivante :

(1) $I(G,H) = n - c(G) + h(G) + f(G)$ **(formule dans [HP95a]).**

Où n est le nombre de gènes communs, $c(G)$ est le nombre de cycles du graphe de points de cassure de G, $h(G)$ est le nombre d'obstacles dans ce graphe et $f(G)$ est à 1 si le graphe représente une forteresse et à 0 sinon.

2.1.3 Résolution des mauvaises composantes

La résolution des mauvaises composantes consiste d'abord à les transformer en de bonnes composantes et ensuite à les résoudre comme l'on résout des bonnes composantes. Afin de transformer ces mauvaises composantes, on dispose de deux opérations, la **fusion** et la **coupure**. La fusion permet de résoudre deux composantes A et B, en effectuant une inversion déterminée par un arc quelconque de A et un arc quelconque de B. De cette façon, ces deux mauvaises composantes sont transformées en une seule composante orientée. La coupure est une inversion effectuée sur

n'importe quels deux arcs noirs d'un même cycle d'une mauvaise composante, ce qui la transforme en une bonne composante.

Lorsque la permutation possède un nombre pair d'obstacles, une fusion est effectuée avec deux obstacles non consécutifs (si possible), c'est-à-dire séparés par au moins un arc noir appartenant à un autre obstacle. Si la permutation contient un nombre impair d'obstacles et qu'il existe au moins un obstacle simple parmi ceux-ci, alors on doit effectuer une coupure sur un de ces obstacles simples. Par contre, s'il n'existe pas d'obstacle simple, nous sommes alors en présence d'une forteresse et l'on doit effectuer une fusion entre n'importe quels deux obstacles (figure 12).

Si $h \neq 0$ **Alors** :
- **Si** h pair **Alors** :
  - **Si** $h = 2$ **Alors** :
    - Faire une **fusion** avec les deux obstacles.
  - **Sinon** :
    - Faire une **fusion** entre deux obstacles non consécutifs.
- **Sinon** :
  - **Si** il existe au moins un obstacle simple **Alors** :
    - Faire une **coupure** dans n'importe quel obstacle simple.
  - **Sinon** : (cas d'une forteresse)
    - Faire une **fusion** entre n'importe quel deux obstacles.

Figure 12 : Procédure pour résoudre les obstacles dans l'algorithme HP ([HP95a]). Le nombre d'obstacles est h.

2.1.4 Résolution des bonnes composantes

Lorsque la transformation des obstacles en bonnes composantes a été effectuée, il ne reste que des bonnes composantes que l'on peut résoudre en ne faisant que des bonnes inversions. Cependant, parmi les bonnes inversions, certaines peuvent mener à la formation de mauvaises composantes et doivent être évitées. Une **inversion sûre** est une bonne inversion qui ne mène pas à la formation d'une mauvaise composante. La distance d'inversion, caractérisée par l'équation (1), ne

peut être obtenue que si l'on effectue exclusivement des inversions sûres pour résoudre les bonnes composantes. La recherche d'une inversion sûre à chaque étape de la résolution est le point critique de l'algorithme HP. Plusieurs auteurs ont explicité différents critères permettant d'identifier une inversion sûre ([BE01], [BH96], [HP95a] et [KST97]). Nous verrons dans la prochaine section comment trouver toutes les inversions sûres à chaque étape.

2.2 Algorithme INV

Le premier algorithme que nous avons utilisé dans cette recherche est l'algorithme INV. Il s'inspire de l'algorithme HP au niveau de la décomposition du graphe de points de cassure et du calcul de la distance d'inversion, mais permet de trouver et parcourir le chemin d'inversion en ayant accès à toutes les inversions sûres après chaque inversion faite. Afin de trouver toutes les solutions à chaque étape de la résolution, il faut trouver toutes les inversions sûres. On appelle « solution » une suite minimale d'inversions qui transforme G en H. Jusqu'à présent, il n'existe pas de méthode directe permettant d'identifier toutes les inversions sûres à partir du graphe de points de cassure. Nous procédons donc de la façon suivante.

En premier lieu, toutes les bonnes inversions sont identifiées. Comme nous l'avons vu plus haut, une bonne inversion est déterminée par deux arcs noirs divergents (voir figure 9). Ensuite, pour chaque bonne inversion, on considère le graphe de points de cassure obtenu après avoir effectué l'inversion. Si ce graphe ne contient pas d'obstacle, alors cette bonne inversion est une inversion sûre.

L'ensemble des solutions possibles que nous considérons ici ne contient que les inversions sûres, une fois les mauvaises composantes résolues. Tel que mentionné par Siepel ([SI02]), il existe également plusieurs solutions possibles pour la résolution des obstacles. Tout d'abord, lors de la fusion de deux obstacles non consécutifs, il peut y avoir plusieurs choix de paires d'obstacles possibles. De plus, pour chacune de ces paires d'obstacles, n'importe quel arc noir de chaque obstacle peut être choisi

pour définir l'inversion. Aussi, lors de la coupure d'un obstacle, n'importe quel cycle de cet obstacle peut être choisi et n'importe quelle paire d'arcs noirs de ce cycle peut définir la coupure. Ainsi, la résolution des obstacles peut donc se faire de plusieurs façons, toutes optimales. En pratique cependant, le nombre d'obstacles est très faible et parmi toutes les composantes des quatre paires de génomes à l'étude, un seul obstacle a été trouvé. Nous ne nous sommes donc pas attardés à trouver toutes les solutions possibles pour la résolution des obstacles.

Complexité en temps :

Soit n, le nombre de gènes communs à deux espèces que l'on compare, la construction du graphe de point de cassure prend donc un temps dans O(n). Dans le pire des cas, ce graphe ne contient qu'une seule bonne composante contenant n arcs noirs. Ainsi, le nombre de bonnes inversions à vérifier est au pire $\binom{n}{2}$, c'est-à-dire en $O(n^2)$. Ainsi, trouver toutes les solutions à chaque étape prend un temps dans $O(n^3)$.

2.3 Algorithme IIS

Le deuxième des deux algorithmes utilisés dans ce travail, est l'algorithme IIS. Cet algorithme permet d'ajouter les fonctionnalités d'insertion et de suppression à l'algorithme INV. Ainsi, lorsque l'on compare deux génomes G et H, on peut tenir compte des gènes propres à G (ensemble A_G), des gènes propres à H (ensemble A_H) et des gènes communs aux deux génomes (ensemble A). Si l'on veut transformer G en H, les gènes dans A_G seront considérés comme des suppressions et les gènes dans A_H comme des insertions.

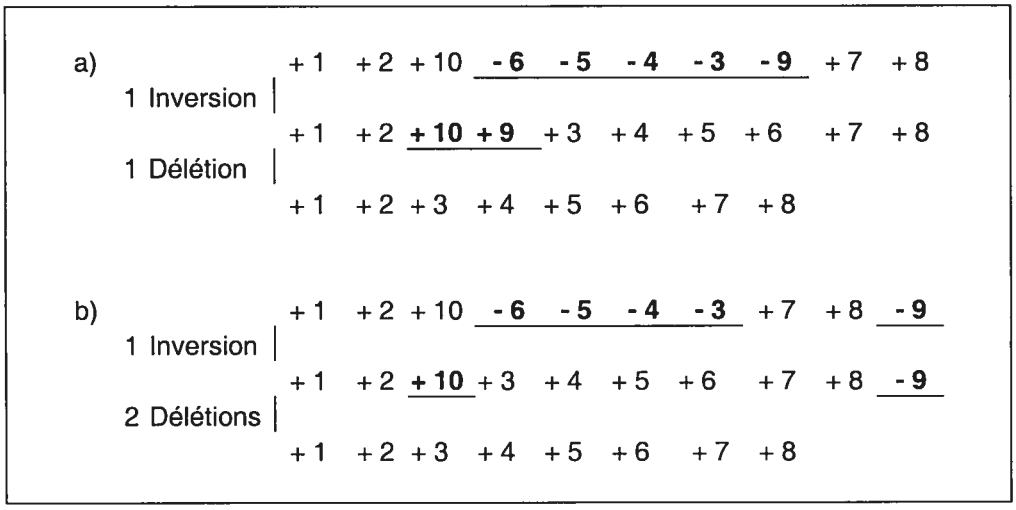

Figure 13 : La différence dans la disposition des gènes propres fait qu'il y a une suppression de plus dans le cas (b). Dans le cas (a), en insérant le gène –9 dans l'inversion, on regroupe les deux gènes à enlever dans un même segment. Ceci n'est pas possible dans le cas (b).

On considère ici l'insertion ou la suppression de blocs de gènes. Ainsi, une opération d'insertion ou de suppression peut correspondre à l'ajout de 10 gènes ou d'un seul, pour autant qu'ils soient contigus dans la permutation. Le nombre d'insertions et de suppressions dépendra alors de la disposition des gènes à enlever et à insérer dans leur génome respectif. Comme le montre la figure 13, nous pouvons voir deux cas où la distance d'inversion entre deux génomes G et H est la même, puisque les gènes communs sont disposés de la même façon dans les deux cas, mais où la dispositions des gènes propres à chaque génome fait que le nombre de suppressions diffèrent entre les deux cas. Nous expliquerons l'algorithme IIS en trois partie : tout d'abord, on ne tient compte que des suppressions, ensuite on ne tient compte que des insertions et finalement, on réunit ces deux aspects pour former l'algorithme IIS.

2.3.1 Les suppressions

Considérons tout d'abord le problème de minimiser les inversions et suppressions. Prenons la transformation de G en H, où l'ensemble A_G est non vide et l'ensemble A_H est vide. Il n'y a donc pas d'insertions à considérer. Pour trouver le chemin d'inversion entre G et H, seuls les gènes communs sont pris en compte et les gènes à supprimer ne sont représentés que par leur position par rapport aux gènes communs. Ainsi, un gène de A_G n'a ni signe, ni numéro, puisque l'ordre et les signes des gènes que l'on supprime n'ont pas d'importance.

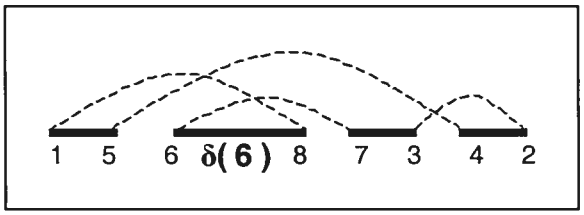

Figure 14 : Graphe de points de cassure incluant un segment de gènes à supprimer sur l'arc (6-8), qui est donc un arc indirect.

Si un ou plusieurs des gènes à supprimer sont entre deux gènes communs x et y d'une permutation v x y z, on utilise le symbole δ(x) pour les représenter tous et la permutation deviendra v x δ(x) y z. Dans le graphe de points de cassure, δ(x) sera l'étiquette de l'arc noir reliant x à y. Un arc noir étiqueté est appelé **arc indirect** et un arc noir non étiqueté est appelé **arc direct** (figure 14). De même, un cycle contenant au moins un arc indirect est appelé **cycle indirect** et une composante comprenant au moins un cycle indirect est appelée **composante indirecte**.

Cette redéfinition des arcs noirs est nécessaire pour définir les inversions qui agissent sur au moins un arc noir indirect. Une inversion est effectuée en brisant un arc noir, et comme les segments à supprimer se trouvent sur les arcs noirs, on peut décider de couper l'arc avant ou après ce segment. Étant donné que le but est de minimiser le nombre de segments à supprimer, le bon choix est de fusionner les segments (figure 15).

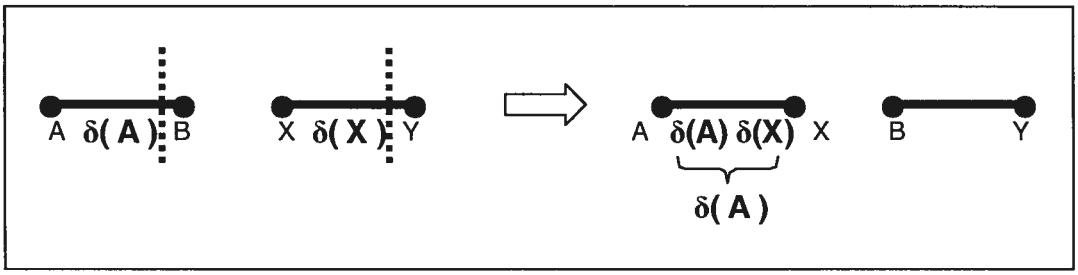

Figure 15 : Inversion causant la fusion de deux segments à supprimer en un seul.

Si l'on prend un cycle orienté de taille n, on peut le résoudre en effectuant (n-1) inversions créant chacune un cycle de taille 1. L'idée est alors d'effectuer chacune de ces inversions de telle sorte que le segment à supprimer ne se retrouve pas dans le cycle de taille 1. Ainsi, peu importe le nombre de segments à supprimer dans le cycle de départ, on peut transformer ce cycle en n cycles de taille 1, dont un seul est indirect ([EM00]). Ceci revient à effectuer une seule suppression à la fin de cette résolution du cycle.

Pour la résolution des obstacles, comme nous l'avons vu plus haut, l'algorithme HP effectue des fusions et/ou des coupures pour transformer ces mauvaises composantes en bonnes composantes. Dans l'optique de réduire les suppressions, les fusions doivent se faire le plus possible entre obstacles indirects, et les coupures à l'intérieur d'un obstacle direct. Ainsi, nous devons redéfinir les opérations de fusion et de coupure des obstacles. Une fusion consiste ici à prendre deux cycles indirects, si possible appartenant à deux obstacles indirects, et si possible non consécutifs. Quand à la coupure, elle est effectuée sur un cycle appartenant si possible à un obstacle simple direct. Les coupures sont effectuées préférablement sur des obstacles directs, afin de garder les obstacles indirects pour des fusions. En effet, les fusions sont plus avantageuses puisqu'elles peuvent regrouper des cycles indirects, ce qui correspond à regrouper des segments à supprimer. Étant donné que l'algorithme HP (figure 11) prévoit des fusions d'obstacles seulement lorsque le nombre d'obstacles est pair, afin de maximiser les fusions, nous devons redéfinir

quelque peu la décomposition du graphe de points de cassure. Il s'agit de regrouper les obstacles indirects consécutifs en un **ensemble indirect** et les obstacles directs consécutifs en un **ensemble direct**.

Tant que il existe un ensemble indirect possédant au moins 3 obstacles **Faire** :
- Une fusion entre deux obstacles non consécutifs de cet ensemble.

{À la fin, tous les ensembles indirects possèdent 1 ou 2 obstacles}

Si il existe au moins 2 ensembles indirects **Alors** :
- **Soit** $I'_1, I'_2, \ldots, I'_R$ la suite des ensembles indirects possédant un seul obstacle.
- **Pour** tous les i appartenant à [1 ; R] **Faire** :
  - Une **fusion** entre le obstacle de I'_i et le obstacle I'_{i+1}.
- **Soit** $I''_1, I''_2, \ldots, I''_S$ la suite des ensembles indirects possédant deux obstacles.
- **Pour** tous les j appartenant à [1 ; S] **Faire** :
  - Une **fusion** entre un obstacle de I''_j et un obstacle de I''_{j+1}.

Figure 16 : Procédure de Fusion d'Obstacles Indirects (FOI) de l'algorithme IIS ([EM00]).

Le graphe peut donc être vu comme une suite $\mathbf{I_1, D_1, I_2, D_2, \ldots, I_N, D_N}$ où consécutifs correspond à une suite maximale d'obstacles indirects non séparés par des obstacles directs, et D_i correspond à une suite maximale d'obstacles directs non séparés par des obstacles indirects. La fusion d'obstacles se fait ensuite selon la procédure de la figure 16 (procédure FOI).

La procédure FOI permet de maximiser le nombre de fusions entre des obstacles indirects. Une fois que les toutes les mauvaises composantes ont été transformées en bonnes composantes, ces bonnes composantes sont résolues en composantes de taille 1, tout en minimisant le nombre de segments à supprimer, comme nous l'avons expliqué plus haut. Ensuite, il suffit d'effectuer la suppression de tous les segments de gènes appartenant à A_G et la transformation de G en H est complète.

Soient H_i le nombre d'obstacles indirects dans la permutation, E_i le nombre d'ensembles indirects et E_d le nombre d'ensembles directs. Alors le nombre $F(H_i)$ de fusions effectuées par la procédure FOI se calcule ainsi :

$F(H_i) = H_i / 2$, si $E_i = 1$, $E_d > 0$ et H_i est pair ,

$F(H_i) = ENT(H_i / 2)$, sinon. ($ENT(...)$ signifiant «arrondi à l'entier inférieur»)

De plus, soit C_i le nombre de cycles indirects de la permutation, alors le nombre de suppressions $S(G,H)$ est :

$S(G,H) = C_i - F(H_i)$.

Finalement, El-Mabrouk prouve dans [EM00] que la distance d'inversions et suppressions $IS(G,H)$, transformant G en H, est donnée par :

$$\boldsymbol{IS(G,H) = I(G,H) + S(G,H)}$$

$$\boldsymbol{= I(G,H) + C_i - F(H_i)} \qquad \textbf{([EM00]).}$$

2.3.2 Les insertions

Prenons maintenant le cas où l'on veut transformer le génome G en H, avec l'ensemble A_G vide et l'ensemble A_H non vide. Ce problème consiste donc à minimiser le nombre d'insertions dans le génome G en tenant compte du nombre minimal d'inversions pour transformer G en H. Ceci équivaut à résoudre le problème de minimiser le nombre de suppression dans le génome H en tenant compte du nombre minimal d'inversions pour transformer H en G. Ce cas peut donc être résolu par l'algorithme d'inversion et suppression, vu dans la partie précédente, avec la différence que l'ordre et le numéro des gènes de A_H dans le génome H doivent être pris en compte.

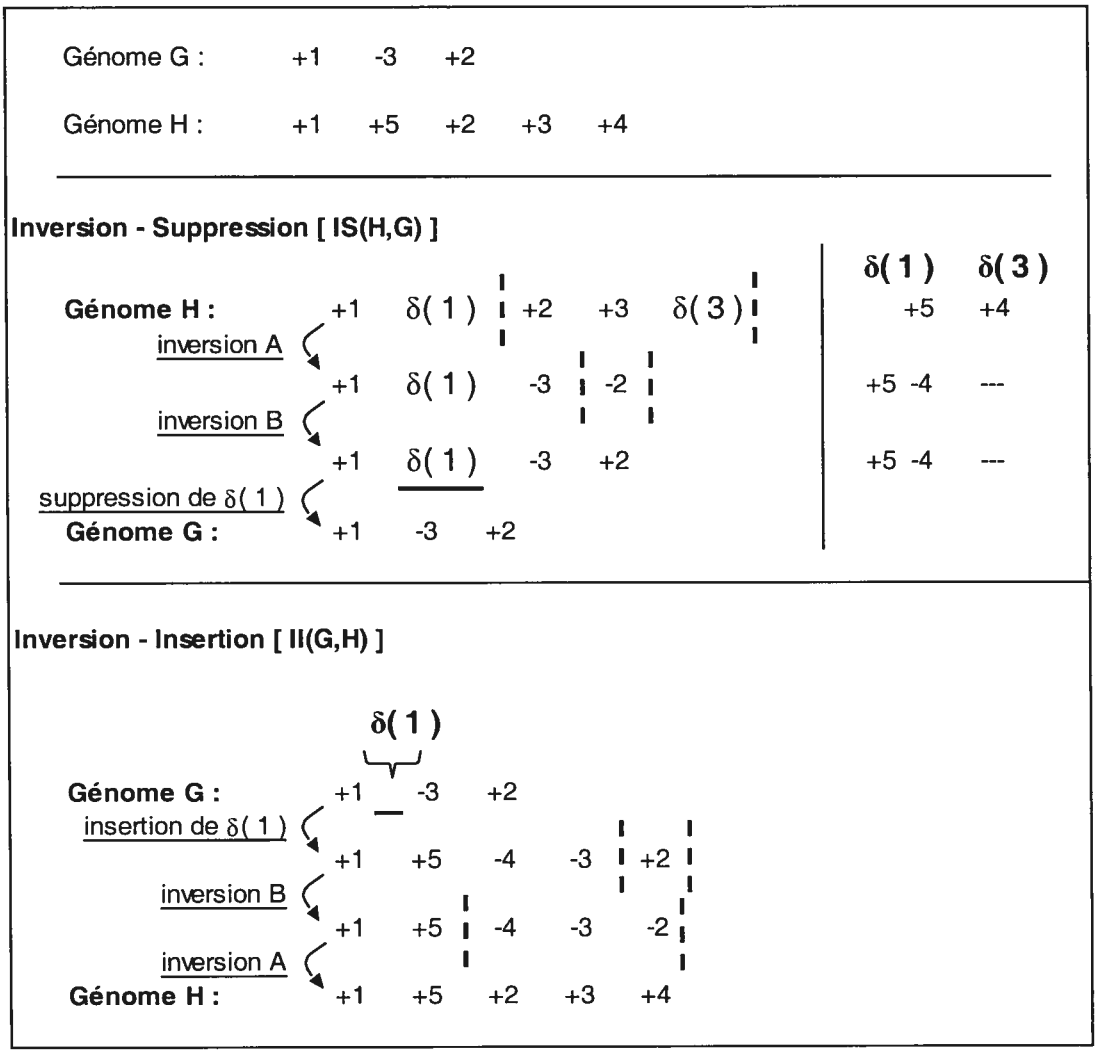

Figure 17 : Exemple de l'équivalence inverse entre la résolution du chemin d'inversion-suppression de H vers G, et la résolution du chemin d'inversion-insertion de G vers H. Les lignes pointillées montrent les emplacements des inversions.

Une fois le chemin d'inversion trouvé et le nombre minimal de segments à supprimer définis, on peut considérer ces segments comme étant des insertions dans le génome G. Comme le montre l'exemple de la figure 17, l'algorithme d'inversion-suppression pour transformer H en G, correspond à faire deux inversions (l'inversion A suivie de l'inversion B) et une suppression du segment δ(1). Inversement,

l'algorithme d'inversion-insertion consiste à faire d'abord une insertion du même segment δ(1) suivie de deux inversions (l'inversion B suivie de l'inversion A).

La distance d'inversion et d'insertion *II(G,H)*, se calcule donc de la même façon que la distance d'inversion et de suppression *IS(G,H)* :

$$II(G,H) = I(H,G) + S(H,G)\ ,$$ **([EM00]).**

2.3.3 Suppression et insertion

Il faut maintenant combiner les deux situations que nous venons de voir, soit la transformation de G vers H, où A_G et A_H sont non vides, par inversions, suppressions et insertions. Nous utiliserons le **génome réduit** de G, noté $\hat{\mathbf{G}}$, qui correspond au génome de G moins les gènes de A_G. De même, le génome réduit de H, noté $\hat{\mathbf{H}}$, correspondant au génome de H moins les gènes de A_H. L'algorithme d'Inversion-Insertion-Suppression (algorithme IIS) est composé de trois étapes principales. La première consiste à trouver le nombre minimal d'insertions à effectuer dans $\hat{G}$ pour former un génome $\hat{G}^C$ contenant tous les gènes de A_H. Pour cela, il suffit d'appliquer l'algorithme d'inversion et suppression aux génomes H et $\hat{G}$. À la fin de l'algorithme, les positions et les segments à supprimer dans H sont interprétés comme les positions et les segments à ajouter dans $\hat{G}$.

À la deuxième étape, on construit le génome G^C en replaçant dans $\hat{G}^C$ les gènes de A_G de façon appropriée. Le génome obtenu contient alors tous les gènes de G et de H. Finalement, la dernière étape consiste à transformer G^C en H par l'algorithme d'inversion et suppression. Les étapes 1 et 3 consistent seulement à appliquer l'algorithme décrit à la section 2.3.1. Nous expliquons maintenant plus en détail l'étape 2.

Une fois la première étape effectuée, nous obtenons le génome $\hat{G}^C$ dans lequel il faut maintenant indiquer la position des éléments de A_G pour obtenir G^C. La position de ces éléments est cependant définie par rapport aux éléments de $\hat{G}$ et non

de $\hat{G}^C$. Ainsi, si un segment de A_G se trouve entre deux gènes A et B de $\hat{G}$, mais que dans $\hat{G}^C$, il existe un gène C entre A et B, nous aurons le choix de placer ce segment soit entre A et C, soit entre C et B. Ce choix sera fait de sorte à minimiser le nombre de suppressions.

Comme on peut le voir dans la figure 18, le graphe de points de cassure de G montre deux segments de A_G positionnés dans deux cycles différents, laissant présumer que deux suppressions seront nécessaires. Par contre, le graphe du génome complété G^C (les gènes 4 et 5 appartenant à A_H) offre deux possibilités de positionnement pour chacun de ces segments. Pour le segment δ(3), les deux possibilités sont équivalentes mais pour le segment δ(6), l'une des deux permet de mettre δ(6) dans le même cycle que δ(3), ce qui permettra de ne faire qu'une suppression. La procédure construction-G^C permet de bien positionner les segments de A_G (figure 19).

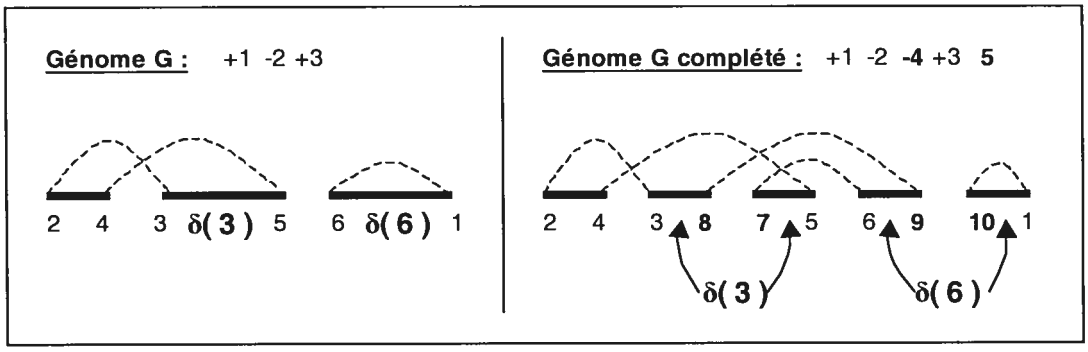

Figure 18 : Possibilités de positionnement des segments de A_G lors de la construction de G^C.

Pour chaque arc noir (a,b) de $\hat{G}$, tel qu'il existe un segment X(a) de A_G dans G **Faire** :

- **Si** a et b sont adjacents dans $\hat{G}^C$ **Alors** :
  - **Transformer** (a,b) en un arc indirect étiqueté X(a).

{Entre chaque arc noir (a,b) de $\hat{G}$, il existe une suite $y_1 \ldots y_p$ de A_H dans $\hat{G}^C$ }

Pour tous les arcs noirs (a,b) restant **Faire** :

- **Si** un des deux arcs (a,y_1) ou (y_p,b) appartient à un cycle indirect **Alors** :
  - **Transformer** cet arc noir en un arc indirect étiqueté X(a).
- **Sinon** :
  - **Si** (a,y_1) ou (y_p,b) appartient à un obstacle indirect **Alors** :
    - **Transformer** cet arc en arc indirect étiqueté X(a).
  - **Sinon** :
    - **Transformer** n'importe lequel de ces deux arcs en arc indirect étiqueté X(a).

Figure 19 : Procédure Construire-G^C de l'algorithme IIS ([EM00]).

Récapitulons maintenant les différentes étapes de l'algorithme IIS permettant de transformer la permutation G en H, où A_G et A_H sont non vides :

- Appliquer l'algorithme INV avec suppressions (partie 2.3.1) pour transformer H en $\hat{G}$. À travers les inversions, on garde le sens et les positions relatives de chaque gène à supprimer. On obtient ainsi le nombre minimum et les positions des segments S_H à insérer dans $\hat{G}$ pour obtenir $\hat{G}^C$.
- Insérer les segments S_H (avec les sens et positions relatives des gènes) dans la permutation $\hat{G}$, pour obtenir ainsi la permutation $\hat{G}^C$, contenant les éléments A_H.
- Positionner les éléments de A_G dans la permutation $\hat{G}^C$, selon la procédure construire-G^C, pour obtenir la permutation G^C. G^C ainsi construite, contient tous les éléments de G en plus des éléments de A_H.
- Appliquer de nouveau l'algorithme INV avec suppressions pour transformer la permutation G^C en H.

Le nombre de réarrangements IIS(G,H) effectué par cette méthode peut se calculer ainsi :

(2) $IIS(G,H) = I(\hat{G},\hat{H}) + S(H, \hat{G}) + S(G^C,H)$ **([EM00]).**

Où $I(\hat{G},\hat{H})$ est le nombre d'inversion minimum calculé selon HP, $S(H,\hat{G})$ est le nombre d'insertions et $S(G^C,H)$ est le nombre de suppressions. La fonction $S(...)$, décrite à la section 2.3.1, est telle que $S(G,H) = C_i - F(H_i)$, où C_i correspond au nombre de cycle de G et $F(H_i)$ est le nombre de fusion de hurdles effectuées.

L'algorithme d'inversion avec uniquement les suppressions est un algorithme exact, c'est-à-dire, qu'il trouve toujours une solution optimale au problème. Par contre, l'algorithme IIS est une heuristique qui est exacte quand le nombre de suppressions pour passer de G^C à H est égal à celui pour passer de G à $\hat{H}$, c'est-à-dire quand $S(G^C,H) = S(G,\hat{H})$. Lorsque ce n'est pas le cas, la solution trouvée pour transformer G en H peut être différente de celle trouvée pour transformer H en G.

Chapitre 3 :

Étude du mécanisme des inversions

3.1 Introduction

L'algorithme INV de la section 2.2, permet de trouver un nombre minimal d'inversions pouvant transformer une permutation en une autre. Bien que cette distance d'inversion soit unique, les chemins d'inversion optimaux ne le sont pas. L'accès à tous les chemins d'inversions optimaux permet de faire ressortir des caractéristiques générales des inversions.

Comme nous l'avons vu au chapitre 1, certains auteurs ([PT03] et [TC00]) ont suggéré que les inversions ne se faisaient pas au hasard, tant au niveau de leur emplacement que de leur taille. Nous nous sommes intéressé à la prédominance des inversions de petites tailles, plus particulièrement aux inversions d'un seul gène. Aussi, nous avons voulu vérifier si l'hypothèse de Tillier et Collins ([TC00]), concernant la symétrie des inversions par rapport à un axe de réplication, pouvait expliquer les résultats que nous avons obtenus.

3.2 Échantillons réels

Nos expérimentations ont été effectuées sur des comparaisons de génomes de bactéries étudiées par Elizabeth Tillier (Université de Toronto). Nous avons utilisé huit génomes de bactéries regroupés par paires : *Chlamydia trachomatis* et *Chlamydophila pneumonia*, *Bacillus halodurans* et *Bacillus subtilis*, *Escherichia coli* et *Vibrio cholerae*, et *Streptococcus pyogenes* et *Staphylococcus aureus*. Nous

rappelons que ces génomes sont des génomes uni-chromosomaux circulaires. Les caractéristiques de ces génomes sont indiquées dans le tableau I.

Paires de génomes	**Tailles des génomes**	**Nombre de gènes communs**	**Distance d'inversion**
Chlamydia trachomatis	894	816	91
Chlamydophila pneumonia	1110		
Bacillus halodurans	4066	2208	979
Bacillus subtilis	4112		
Escherichia coli	4279	1648	717
Vibrio cholerae	2742		
Streptococcus pyogenes	1845	939	649
Staphylococcus aureus	2714		

Tableau I : La taille indique le nombre de gènes identifiés et les gènes communs sont les gènes homologues identifiés dans les deux génomes. La distance d'inversion est obtenue par l'algorithme INV en ne tenant compte que des gènes communs.

3.3 Méthodologie

À chaque étape des algorithmes INV et IIS, toutes les inversions sûres sont retrouvées. Nous effectuons alors une sélection parmi celles-ci. Cette sélection, que nous allons appeler **sélection prioritaire**, consiste à prendre toutes les inversions sûres, à les trier selon les critères établis, et à effectuer celle répondant le plus à ces critères. Les caractéristiques que nous avons retenues sont au niveau de la taille et de la position des inversions.

Dans un premier temps, aucune contrainte de taille n'a été imposée pour les inversions. Autrement dit, à chaque étape, une inversion sûre est choisie de façon aléatoire. Dans un deuxième temps, nous avons voulu favoriser les petites inversions. Pour ce faire, à chaque étape, une inversion sûre a été choisie aléatoirement parmi les plus courtes. Les résultats obtenus sur les paires de génomes réels ont été comparés avec ceux obtenus sur des permutations aléatoires.

Pour chaque paire de génomes réels, nous avons construit des génomes aléatoires ayant un nombre de gènes et une distance d'inversion à la permutation identité équivalents à ceux des génomes réels. Les inversions ont été effectuées de façon aléatoire tant au niveau de la taille que de l'emplacement. Les distances d'inversion obtenue pour ces génomes aléatoires ne sont pas toujours exactement les mêmes que pour le génome réel, en effet, plus le rapport *nombre de gènes / distance d'inversion* diminue, moins le génome aléatoire avait la bonne distance d'inversion. Nous avons accepté des différences de moins de 10% de la distance d'inversion originale. De plus, cinq réplicats de génomes aléatoires ont été construits pour chaque paire de génome réel à l'étude.

L'algorithme INV a d'abord été utilisé. Le calcul de la taille des inversions a inclus les gènes communs et les segments de gènes à supprimer situés entre les deux arcs noirs définissant l'inversion. L'algorithme IIS a ensuite été utilisé. Dans ce cas, le calcul de la taille des inversions a inclus les gènes communs, les gènes insérés de H et les segments de gènes de G à supprimer et situés entre les deux arcs noirs définissant l'inversion. Pour les deux algorithmes, les programmes ont été exécutés cinq fois sur chaque paire de génomes et pour chaque type de sélection des inversions.

3.4 Les inversions de taille 1

Les figures 20 et 21 représentent les résultats obtenus par notre algorithme d'inversion et suppression. La figure 20 correspond au choix aléatoire des inversions

sûres et la figure 21 correspond à la sélection prioritaire des plus petites inversions sûres. Les courbes correspondent au nombre moyen d'inversions obtenues en fonction de la taille des inversions. Les données pour les paires d'espèces sont représentées par les courbes pleines et bleues, alors que les données des permutations aléatoires équivalentes sont représentées par les courbes pointillées et rouges. La colonne de gauche montre les résultats totaux, les tailles étant arrondies à 20 gènes, alors que la colonne de droite ne montre que les fréquences des inversions de taille inférieure ou égale à 20 gènes, les tailles n'étant pas arrondies.

Dans le figure 20, on remarque que trois des quatre espèces (toutes sauf *S.aureus* / *S. pyogenes*) indiquent un nombre plus élevé de petites inversions par rapport aux permutations aléatoires. En regardant le détail des tailles inférieures à 20 gènes, on remarque plus spécifiquement que ce sont les inversions de taille 1 qui sont en plus grand nombre.

Pour avoir une meilleure idée de la prédominance des inversions de taille 1, nous avons effectué la même expérience, mais en favorisant les petites inversions. Les résultats sont présentés à la figure 21. On y remarque un accroissement des inversions de petites tailles, ce qui était prévisible. Plus intéressants, les résultats pour les espèces montre une grande accentuation des inversions de taille 1, et ce même pour la paire d'espèces *S.aureus* / *S. pyogenes*.

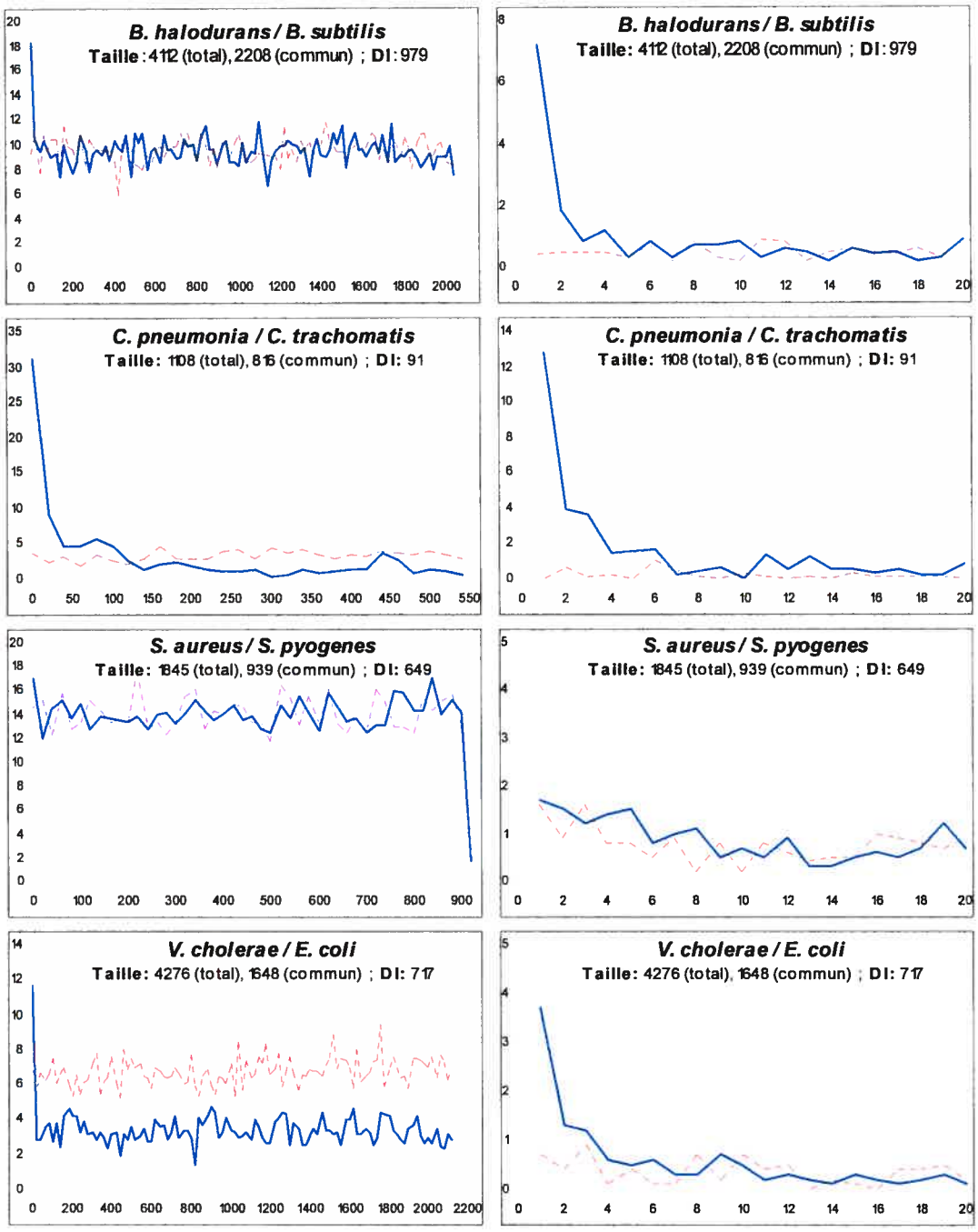

Figure 20 : Résultats de l'algorithme INV avec un choix aléatoire des inversions, pour les quatre paires d'espèces à l'étude. Les noms d'espèces, les tailles des génomes (total et commun) et la distance d'inversion (DI), sont en titre. Les courbes montrent le nombre d'inversions (en ordonnée) en fonction de la taille des inversions (en abscisse). Les courbes bleues sont les données des génomes réels et les courbes rouges sont les données des génomes aléatoires correspondant.

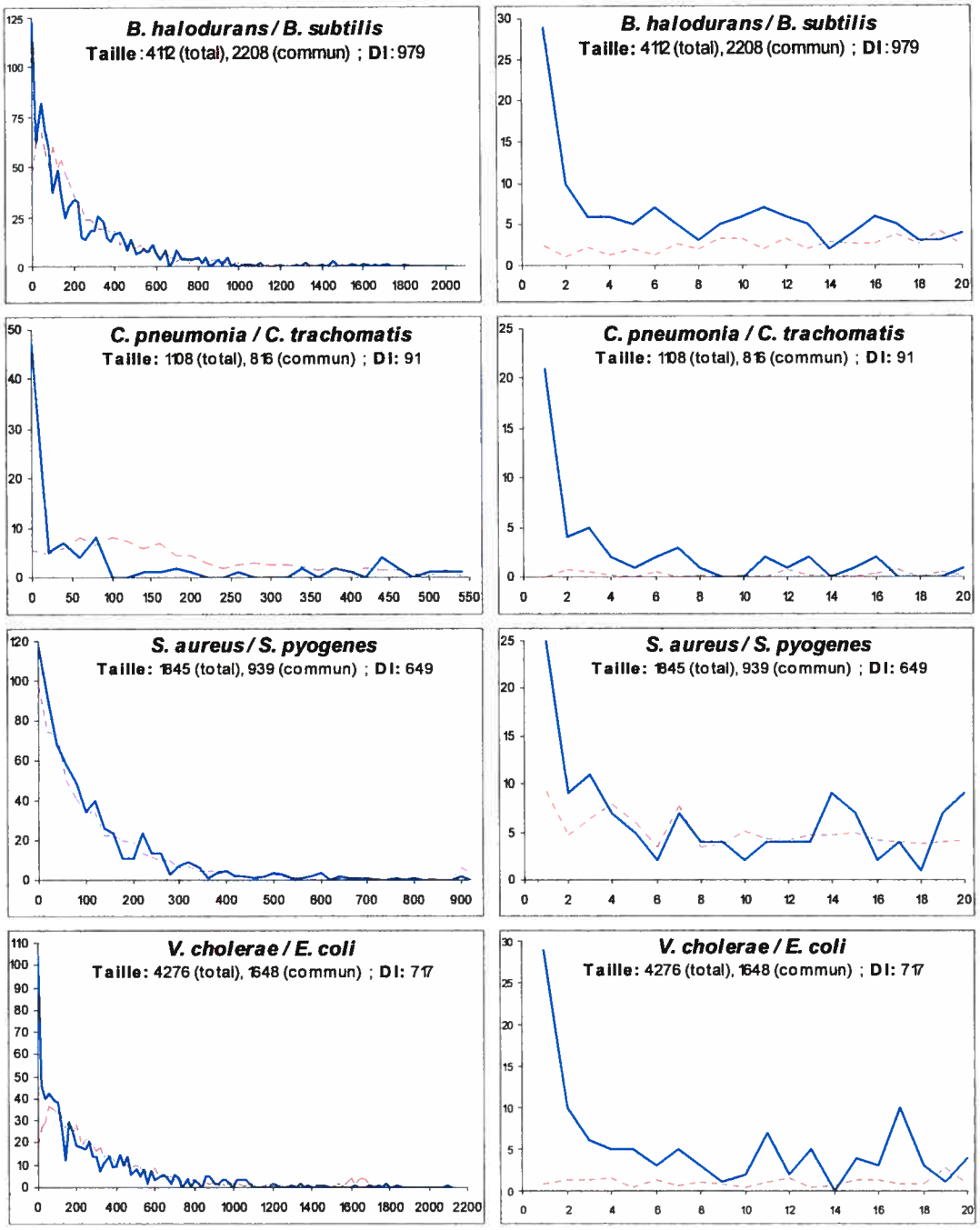

Figure 21 : Résultats de l'algorithme INV avec un choix des plus petites inversions, pour les quatre paires d'espèces à l'étude. Les noms d'espèces, les tailles des génomes (total et commun) et la distance d'inversion (DI), sont en titre. Les courbes montrent le nombre d'inversions (en ordonnée) en fonction de la taille des inversions (en abscisse). Les courbes bleues sont les données des génomes réels et les courbes rouges sont les données des génomes aléatoires correspondant.

Ces deux séries de tests permettent donc de déduire qu'il y a plus d'inversions de taille 1 parmi les inversions sûres pour les espèces à l'étude. Ce nombre élevé d'inversions de taille 1 peut s'expliquer de plusieurs façons. La plus évidente est que dans la suite de gènes d'un génome, on ait « $+a$ $+b$ $+c$ » et que dans l'autre, on ait « $+a$ $-b$ $+c$ ». Dans ce cas, il est évident que seule une inversion de taille 1 peut transformer l'un en l'autre. Dans le cas où il y aurait un segment à supprimer avant et/ou après le gène *b* (gènes présents dans un génome mais pas dans l'autre), il se peut que l'inversion de taille 1 identifiée soit, en fait, une plus grande inversion incluant les gènes supplémentaires. Ces deux cas peuvent être facilement détectés, par simple observation des séquences génomiques.

Il est également possible qu'une inversion de taille 1 soit la conséquence d'un autre type de réarrangement. En effet, si l'on prend le cas d'une transposition du gène *b*, la suite de gènes passerait de « $+a$ $+b$ $+c$ ... $+u$ $+v$ » à « $+a$ $+c$ ...$+u$ $+b$ $+v$ ». Ainsi, il faudrait trois inversions pour retrouver la bonne suite, soit une entre *c* et *b* pour ramener le gène *b* au côté de *a*, une entre *c* et *u* pour ramener *c* au côté de *b*, et une dernière de taille 1 pour mettre *b* dans le bon sens. De même que précédemment, des segments à supprimer peuvent se glisser avant et/ou après le gène b et l'inversion apparente de taille 1 peut en fait correspondre à une plus grande inversion.

Il est également possible que des inversions apparaissant de taille 1 soient, en fait, la conséquence d'une erreur d'identification des orthologues. Ce type d'erreur peut venir du fait que l'on se soit trompé de duplicat, c'est-à-dire qu'on ait choisit le mauvais duplicat comme gène orthologue.

Afin de savoir à quel point ces cas représentent nos données, nous avons retracé les gènes ayant été impliqué dans une inversion de taille 1 lors de la sélection prioritaire des inversions de petites tailles, et nous avons analysé leur position dans la séquence génomique de départ. Les résultats de cette analyse sont donnés dans le tableau II. Les deux premières colonnes concernant le positionnement du gène *b*, correspondent à des cas où les inversions de taille 1 que nous avons trouvé sont

clairement dues à des changements d'orientation de gènes singuliers. Les deux dernières colonnes indiquent les cas où nous avons trouvé une inversion de taille 1, mais où il y a possibilité qu'un autre type de réarrangement ou une erreur d'identification, se soit produit.

Paires d'espèces	Inversions de taille 1	+a –b +c	+a *I* –b *I* +c	u b v	u *I* b *I* v
C. trachomatis / C. pneumonia	21	14	4	1	2
B. halodurans / B. subtilis	29	8	4	7	10
E. coli / V. cholerae	29	8	8	6	7
S. pyogenes / S. aureus	25	2	4	12	7

Tableau II : Répartition des inversions de taille 1 selon leur positionnement dans la séquence génomique de départ. *+a-b+c* indique un gène dans le même contexte dans les deux génomes. Le cas *ubv* indique que le gène *b* n'est pas entouré par les mêmes gènes dans les deux génomes. *I* indique des insertions de gènes.

3.5 Insertions / Suppressions

La même expérience a été faite avec l'algorithme d'inversion, insertion et suppression. Les résultats retenus sont ceux transformant le génome complété de G (G^C) en H. Ils sont présentés dans les figures 22 et 23, où la figure 22 représente les résultats obtenus en choisissant les inversions sûres de façon aléatoire, et la figure 23 représente ceux obtenus en choisissant prioritairement les petites inversions. Concernant le cas où les inversions sont choisies aléatoirement (figure 22), nous remarquons que le nombre d'inversions de taille 1 est plus grand dans le cas de *S.aureus / S. pyogenes* et *V. cholera / E. coli*, qu'il est équivalent pour *C.pneumonia /*

C. trachomatis et qu'il diminue pour *B. subtilis / B.halodurans*. Étant donné que les distances d'inversion sont les mêmes avec ou sans insertions/suppressions, nous pouvons comparer les résultats de façon absolue. Cependant, si l'on regarde la tendance générale du nombre de petites inversions, nous retrouvons le même scénario que dans la première expérience, avec les mêmes trois espèces contre une.

En effet, bien que *S.a. / S.p.* ait un nombre plus grand d'inversions de taille 1, cela ne représente pas une tendance comparativement aux autres tailles d'inversions. Par contre, pour *B.s. / B.h.*, qui retourne plus d'inversions de taille 2 que de taille 1, il existe tout de même une forte tendance aux petites inversions. Dans ce dernier cas, on pourrait dire que les insertions ont causé un léger décalage de la courbe vers la droite. D'ailleurs, pour toutes les espèces, les tailles 2 à 5 ont augmenté en nombre comparativement aux résultats pour l'algorithme d'inversion et suppression.

Concernant les résultats avec choix prioritaire des petites inversions (figure 23), les courbes montrent deux particularités intéressantes. La première est que trois des quatre courbes possèdent deux pics (l'exception étant *C.p. / C.t.*), un premier étant pour les tailles de 0 à 20 et un deuxième pour les tailles d'environ 10% de la taille maximale des inversions. De plus, bien qu'elles forment un pic, les inversions de tailles 1 ne sont pas nécessairement les plus nombreuses. La deuxième particularité est que le nombre des inversions de taille 0 à 20 est de beaucoup plus faible que ce que montre les résultats sans insertions (figure 21), la plus grande baisse étant pour *B.h. / B.s.* où elles passent d'environ 125 à 16.

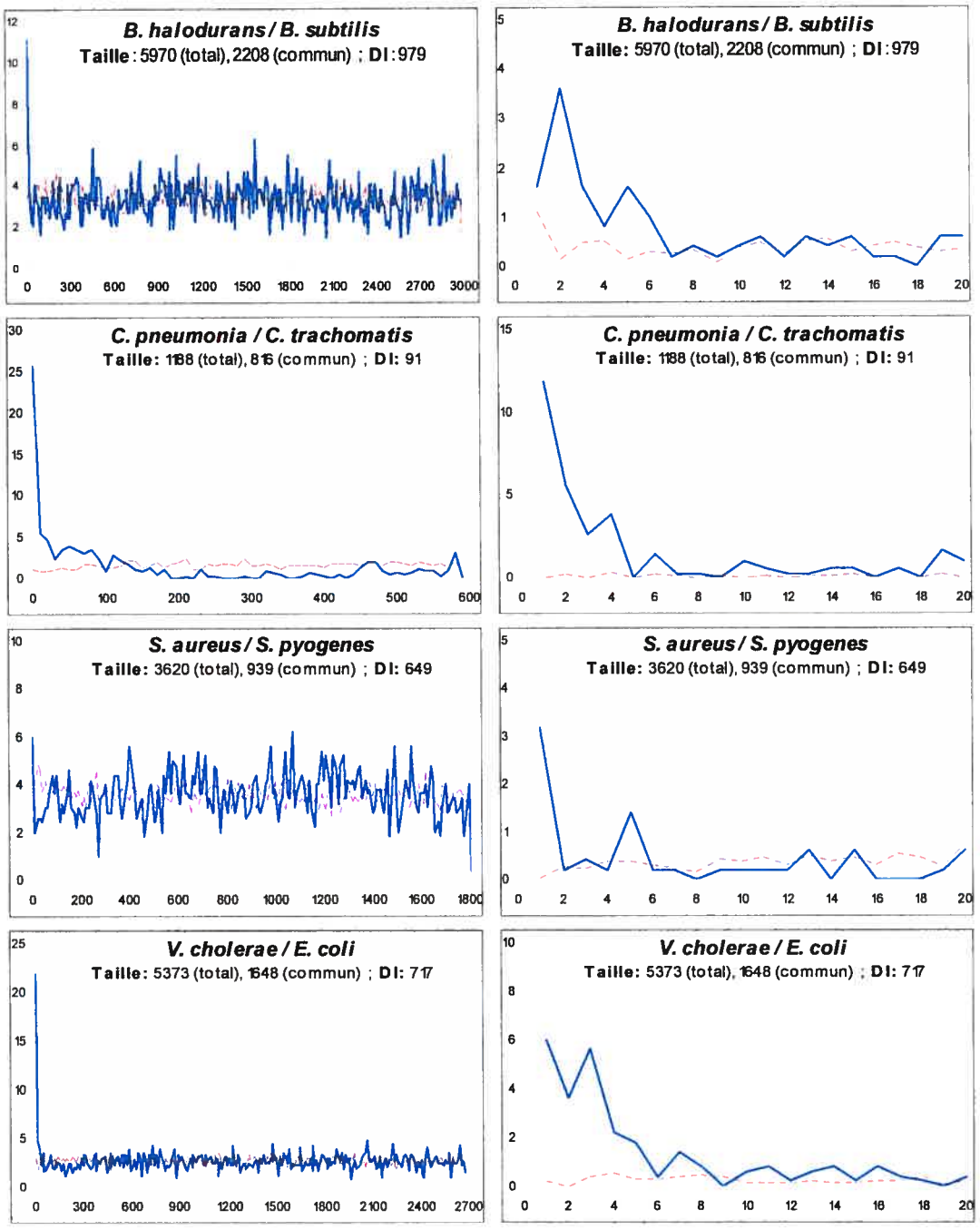

Figure 22 : Résultats de l'algorithme IIS avec un choix aléatoire des inversions, pour les quatre paires d'espèces à l'étude. Les noms d'espèces, les tailles des génomes (total et commun) et la distance d'inversion (DI), sont en titre. Les courbes montrent le nombre d'inversions (en ordonnée) en fonction de la taille des inversions (en abscisse). Les courbes bleues sont les données des génomes réels et les courbes rouges sont les données des génomes aléatoires correspondant.

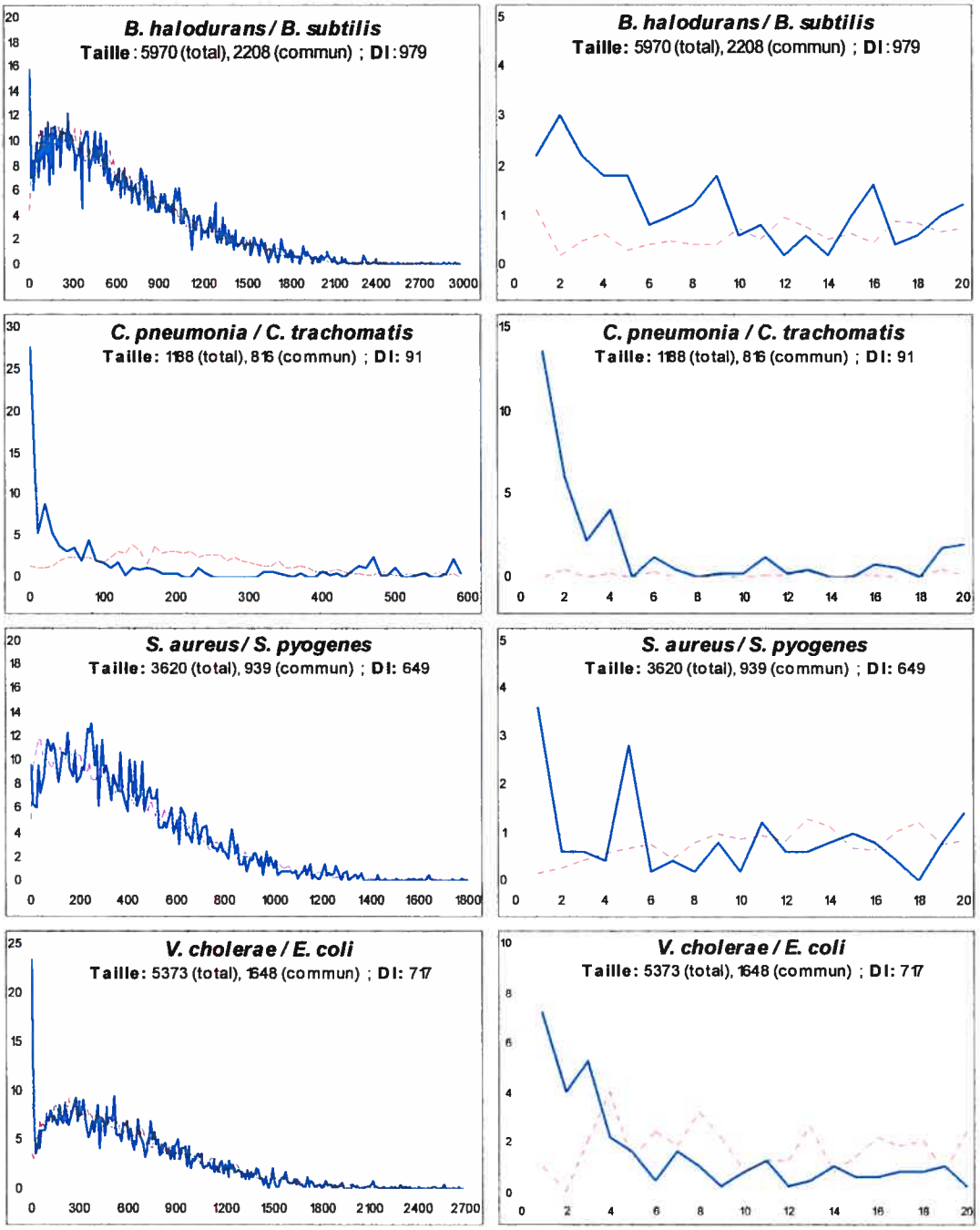

Figure 23 : Résultats de l'algorithme IIS avec un choix des plus petites inversions, pour les quatre paires d'espèces à l'étude. Les noms d'espèces, les tailles des génomes (total et commun) et la distance d'inversion (DI), sont en titre. Les courbes montrent le nombre d'inversions (en ordonnée) en fonction de la taille des inversions (en abscisse). Les courbes bleues sont les données des génomes réels et les courbes rouges sont les données des génomes aléatoires correspondant.

Tout d'abord, comme la figure 23 l'indique, la taille des génomes avec insertions (total) diffère considérablement de la taille des génomes sans insertions (commun). Le nombre de gène total étant de 2,5 à presque 4 fois le nombre de gènes communs. Ce grand nombre de gènes insérés a fait grandir la taille moyenne des inversions. D'ailleurs, la seule paire d'espèces ayant le plus petit nombre de gènes non communs (*C.p. / C.t.*) est la seule paire d'espèces ayant un seul pic pour le nombre d'inversion de taille 1. Il est donc clair que les pics uniques pour les tailles 0 à 20 de la figure 21 se sont divisés en deux pics à cause des gènes insérés.

En ce qui concerne le nombre très inférieur d'inversions de taille 1, il faut tout d'abord noter que les génomes sont beaucoup plus grands alors que le nombre d'inversions est identique. Ainsi, le nombre d'inversions par intervalle de tailles ne peut que diminuer. Il est normal d'observer une baisse pour chacun de ces intervalles. En ajoutant la conséquence de l'insertion d'un grand nombre de gènes que nous venons de mentionner, il est compréhensible que le nombre d'inversions de taille 1 soit beaucoup plus faible lorsque l'on inclut les insertions.

Malgré ces remarques, il reste vrai que le nombre d'inversions de taille 0 à 20 est élevé et c'est grandement dû aux inversions de taille 1, à l'exception d'une seule espèce (*B.h. / B.s.*). Ce phénomène apparaissant donc autant avec ou sans les insertions, nous croyons qu'il s'agit d'une caractéristique importante du mécanisme des inversions chez les bactéries.

3.6 Pertinence des chemins d'inversion obtenus

Plus une distance d'inversion est élevée, moins il y a de chance que l'on retrouve exactement le chemin d'inversion effectué. Il en va de même avec la taille des inversions trouvées. Ainsi, on peut se demander à quel point les résultats que nous avons trouvés, concernant la taille des inversions, reflètent la réalité.

Afin de connaître l'efficacité de notre algorithme à retrouver correctement les tailles d'inversions, nous avons effectué l'expérience suivante : sur un génome de 1000 gènes, nous avons effectué N inversions de taille fixe T, positionnées aléatoirement dans le génome. Les tailles testées ont été T = [5; 10; 15; 20; 50; 100; 200], alors que les nombres d'inversions N ont varié selon les résultats de l'algorithme. Pour chaque T et chaque N, nous avons calculé la proportion de bonnes inversions dans le chemin d'inversion retourné par notre algorithme, une bonne inversion étant une inversion de taille T. Nous avons testé les résultats obtenus en se basant sur deux paramètres ***r*** et ***s***, où :

r est une distance d'inversion telle que *r-2*, *r-1*, *r*, *r+1* et *r+2* sont les cinq distances d'inversion consécutives les plus petites où l'algorithme INV retourne au moins 10 % d'inversions de mauvaise taille.

s est une distance d'inversion telle que *s-2*, *s-1*, *s*, *s+1* et *s+2* sont les cinq distances d'inversion consécutives les plus petites où l'algorithme INV retourne au moins 90 % d'inversions de mauvaise taille.

L'algorithme avec choix aléatoire des inversions (figure 24) et l'algorithme avec choix prioritaire des petites inversions (figure 25), ont été testés.

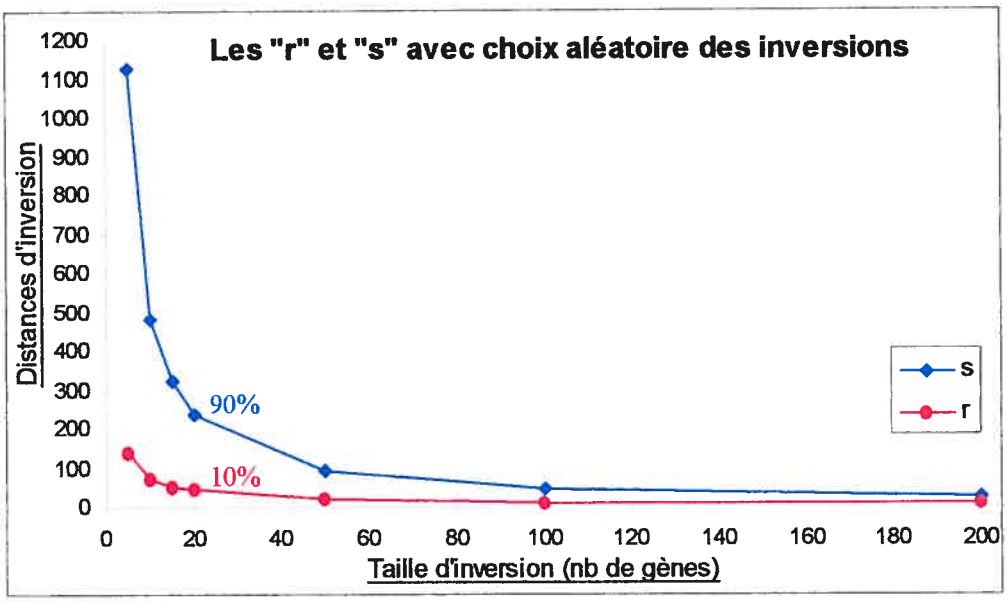

Figure 24 : Les *r* (ligne rouge) et les *s* (ligne bleue) trouvés pour différentes tailles d'inversions faites aléatoirement sur un génome de 1000 gènes.

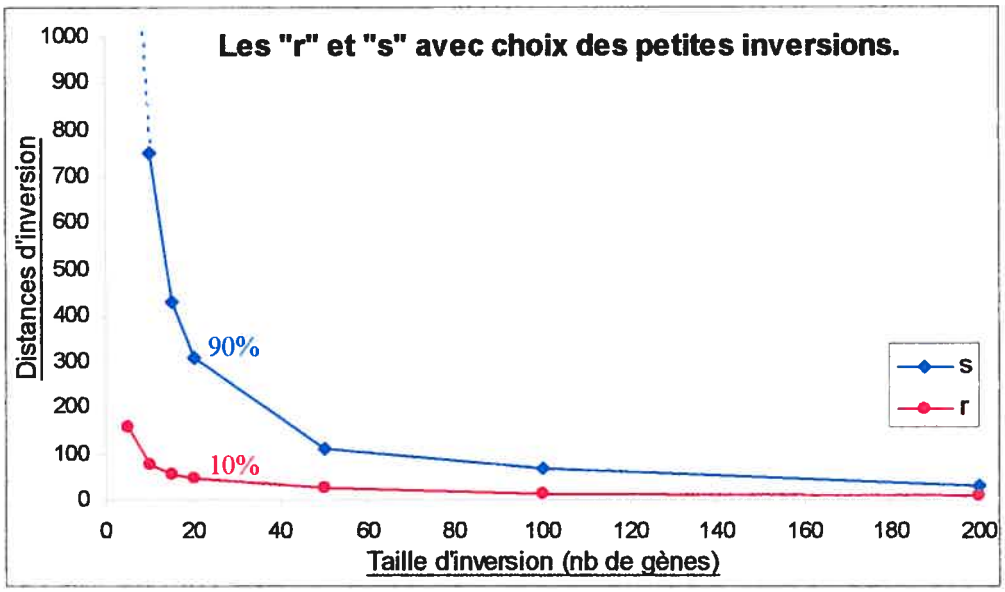

Figure 25 : Les *r* (ligne rouge) et les *s* (ligne bleue) trouvés pour différentes tailles d'inversions faites aléatoirement sur un génome de 1000 gènes. Le choix des petites inversions a été prioritaire. Le résultat pour le *s* de la taille d'inversion 5 est de plus de 1500.

On voit donc de façon très nette que pour des petites inversions, la distance d'inversion doit être très grande avant d'avoir un taux d'erreur significatif. Cependant, le taux d'erreur croît de façon exponentielle en fonction de la taille des inversions. Ainsi, les résultats que notre algorithme retourne sont fiables pour les petites inversions. Il est par contre, très hasardeux d'utiliser les résultats obtenus pour les grandes inversions, afin d'émettre des hypothèses biologiques. En particulier, il n'est pas possible de déduire des résultats biologiques à partir de la tendance uniforme observée à la figure 22, de la répartition des grandes inversions par rapport à leur taille.

3.7 Axe de réplication

Comme nous l'avons mentionné lors de la revue de littérature, Tillier et Collins ([TC00]) ont avancé l'hypothèse que les inversions se font de façon préférentielle autour de l'axe de réplication qui coupe virtuellement en deux le génome circulaire (figure 26). Ils ont remarqué que la position des gènes communs dans un génome était soit approximativement la même, soit diamétralement opposée par rapport à cet axe de réplication. De plus, dans ce deuxième cas, le signe des gènes était inversé. Ce processus laisse à penser que d'apparentes transversions sont en fait deux inversions consécutives de tailles différentes.

L'exemple de la figure 26 montre un segment de quatre gènes inversés dont deux gènes sont d'un côté de l'axe et les deux autres gènes sont de l'autre. Le **coefficient de symétrie**, dénoté C, d'une inversion est : $C = N_1/N_2$, où N_1 est le nombre de gènes inversés d'un côté de l'axe et N_2 est celui de l'autre côté de l'axe. N_1 et N_2 sont tels que $N_1 \leq N_2$. Lorsque C est égal à 1, comme c'est le cas dans l'exemple de la figure 26, l'inversion est parfaitement symétrique par rapport à l'axe, et lorsqu'il est égal à 0, l'inversion ne coupe pas du tout l'axe.

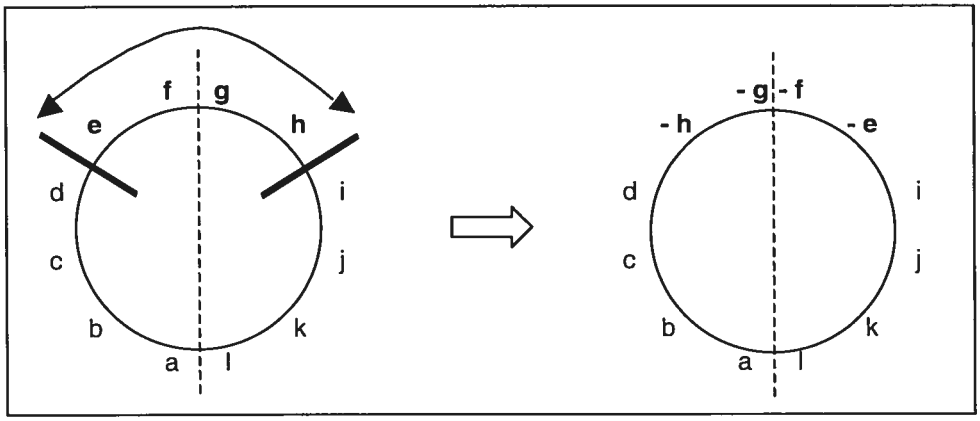

Figure 26 : Exemple d'une inversion symétrique autour de l'axe de réplication (ligne pointillée).

L'expérience effectuée dans l'article de Ajana *et al.* ([ALT02]) a consisté, en partie, à trouver des chemins d'inversions en favorisant les inversions ayant les plus grands coefficients. La comparaison entre les résultats obtenus pour différentes paires d'espèces de bactéries et ceux obtenus pour des permutations aléatoires de taille et de distance d'inversions équivalentes, a confirmé l'hypothèse de [TC00] pour certaines paires d'espèces.

Nous avons vérifié si cette hypothèse pouvait expliquer le nombre élevé des petites inversions apparentes que l'on a trouvé. En effet, comme il est montré à la figure 27, une succession d'inversions symétriques et d'inversions non symétriques pourrait occasionner l'isolation de gènes singuliers dans une séquence génomique. Une inversion de taille 1 devient alors nécessaire pour replacer ces gènes dans la bonne orientation.

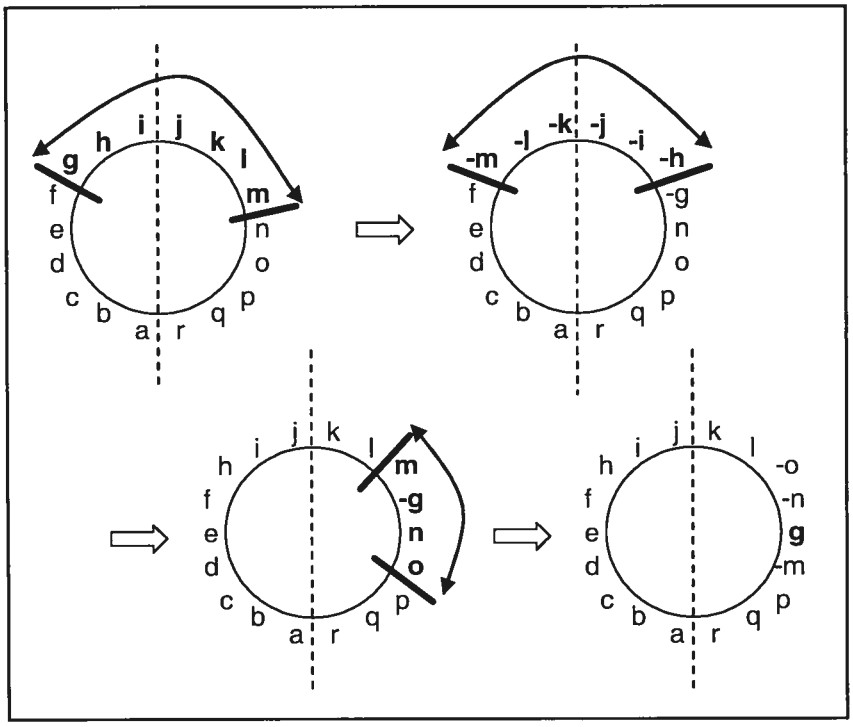

Figure 27 : Exemple de trois inversions consécutives telles que les deux premières sont symétriques et la dernière est totalement asymétrique. La conséquence est la même que lorsque l'on effectue une transposition du gène « g ».

Nous avons donc construit cinq permutations en effectuant des inversions de taille aléatoire mais toutes symétriques à un axe de réplication. Nous avons ensuite construit cinq autres permutations en effectuant au hasard des inversions soit symétriques, soit totalement asymétriques. Une inversion **symétrique** a été déterminée par un coefficient de symétrie $C \geq 0,9$, et l'asymétrie totale correspond à $C = 0$. L'algorithme d'inversion et suppression a ensuite été utilisé, en sélectionnant de façon prioritaire les inversions de petites tailles, pour transformer ces permutations en la permutation identité.

Les résultats sont présentés à la figure 28. Pour une même taille de permutation (1000 gènes), nous avons effectué 500 et 1000 inversions afin de voir si une tendance pouvait se dessiner avec l'accroissement du nombre d'inversions. On peut cependant remarquer que dans aucun des cas, le nombre d'inversion de taille 1 prédomine. Si l'hypothèse de Tillier et Collins avait pu expliquer une plus forte présence de ces inversions, on l'aurait remarqué pour les permutations construites avec des inversions symétriques et asymétriques.

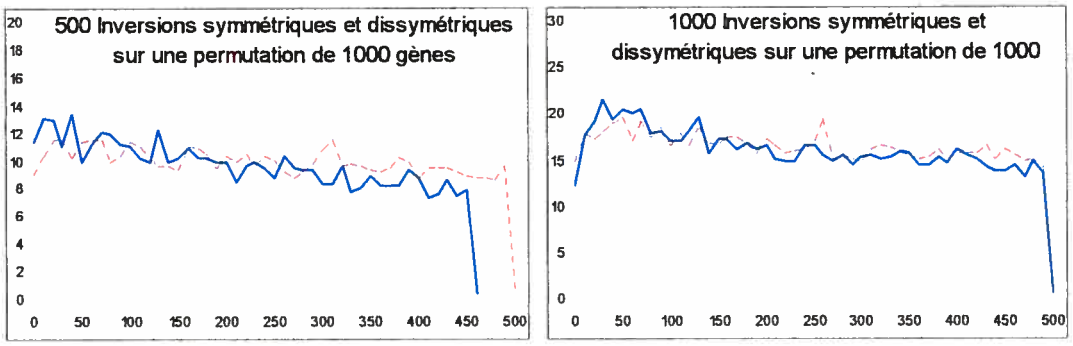

Figure 28 : L'expérience a été effectuée sur des permutation de 1000 gènes construites en effectuant 500 (figure gauche) et 1000 (figure droite) inversions de deux types. Le premier type (courbe pleine bleue) est uniquement symétrique (+/-10%) autour de l'axe et le deuxième type (courbe pointillée rouge) est au hasard (probabilités égales) symétrique ou totalement asymétrique. La courbe indique le nombre d'inversions (ordonnée) en fonction de leur taille en nombre de gènes (abscisse).

Cette dernière expérience ne confirme ni n'infirme l'hypothèse de Tillier et Collins ([TC00]), mais elle montre plutôt qu'on ne peut faire de lien entre cette hypothèse et le nombre élevé d'inversion de taille 1 que l'on a observé.

Conclusions

Grâce à l'augmentation du nombre de génomes complètement séquencés, l'étude des mécanismes évolutifs à l'échelle génomique, devient possible. La génomique comparative fait partie intégrante de cette étude, à travers le calcul de distance d'évolution permettant ultimement la construction d'un arbre phylogénétique des espèces. La présente recherche s'inscrit dans ce cadre d'étude. Nous nous sommes plus particulièrement intéressés à des algorithmes permettant de retrouver des chemins d'inversion optimaux permettant d'expliquer l'évolution de deux génomes. Dans le cas où les deux génomes ne contiennent pas les mêmes gènes, il faut également considérer les insertions et les suppressions de blocs de gènes. Ces méthodes nous ont permis d'étudier les mécanismes d'inversions et d'en déduire des particularités au niveau de la taille et de l'emplacement des inversions dans les génomes à l'étude.

Nous avons généralisé l'algorithme développé par Hannenhalli et Pevzner ([HP95a]) en y ajoutant la fonctionnalité de pouvoir accéder à tous les chemins d'inversions optimaux. Cette fonctionnalité nous a permis de choisir, à chaque étape de l'algorithme, le type d'inversion à effectuer. Nous avons également considéré l'algorithme développé par El-Mabrouk ([EM00]), qui étend l'algorithme HP à la distance d'inversion, d'insertion et de suppression. De ce fait, il a été possible de travailler avec tous les gènes de deux génomes comparés plutôt qu'avec les gènes communs seulement. Deux types de tests ont été effectués avec ces deux algorithmes : le premier consistait à laisser au hasard le choix du type d'inversion à effectuer, et le deuxième à choisir de façon prioritaire les inversions de petite taille.

La comparaison entre quatre paires de génomes de bactéries a ensuite été effectuée. La distance d'inversion et la répartition du nombre d'inversions par

rapport à leur taille en nombre de gènes, ont été calculé avec les deux types de roulements des deux algorithmes, pour chaque paire de génomes. Afin de valider les résultats, nous avons également utilisé des génomes aléatoires équivalents en taille et en distance d'inversion. Les résultats obtenus ont montré un excès d'inversions de taille 1, c'est-à-dire d'inversions d'un seul gène. Cette particularité a été perçue pour les deux algorithmes, quoique de façon moins importante pour l'algorithme incluant les insertions et suppressions.

En observant les génomes à l'étude, on constate qu'une proportion importante de ces inversions correspond, de façon évidente, à des changements d'orientation de gènes singuliers. Cependant, cette situation ne s'applique pas à tous les gènes ayant été inversés. Une explication biologique à cette situation, est qu'un autre type de réarrangement se soit produit. En effet, la transversion d'un seul gène peut expliquer l'isolement d'un gène de sa suite, ainsi que son sens inverse. Une autre possibilité serait que des erreurs se soient glissées lors de l'identification des gènes orthologues. On pourrait alors supposer que l'emplacement du gène orthologue véridique ne causerait pas une inversion de taille 1. Cependant, les erreurs possibles d'identification ne peuvent, à elles seules, expliquer les résultats obtenus. Bien que nous ne puissions expliquer la cause exacte de toutes ces inversions de taille 1, il n'en reste pas moins que nos résultats reflètent un mécanisme biologique particulier qui est le déplacement de gènes uniques à travers le génome.

Nous avons également vérifié si une hypothèse émise par Tillier et Collins ([TC00]), concernant la symétrie des inversions par rapport à un axe de réplication, pouvait expliquer cette présence d'inversions de taille 1. Nos résultats n'ont cependant montré aucun lien direct entre la symétrie des inversions par rapport à un axe de réplication, et le nombre élevé d'inversions de taille 1 retourné par nos algorithmes.

Pour complètement comprendre ce que nous avons observé, d'autres recherches devront être effectuées tant au niveau de l'identification des gènes

orthologues dans les génomes, qu'au niveau de l'étude d'autres types de réarrangements comme mécanismes évolutifs.

Bibliographie

[ALT02] Y. Ajana, J-F Lefebvre, E.R.M. Tillier et N. El-Mabrouk. 2002. Exploring the set of all minimal sequences of reversals – An application to test the replication-directed reversal hypothesis in bacteria. *Proceedings of the Second International workshop on Algorithms in Bioinformatics (WABI 2002)*, edited by R. Guigo and D. Gusfield. *Lecture notes in Computer Science*, vol. 2542: 300-315.

[BCH02] A. Bergeron, C. Chauve, T. Hartman et K. St-Onge. 2002. On the properties ok sequences of reversals that sort a signed permutation. *Proceedings of JOBIM'02*, 99-108

[BE01] A. Bergeron. 2001. A very elementary presentation of the Hannenhalli-Pevzner theory. *12th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science*, 106-117.

[BH96] P. Berman et S. Hannenhalli. 1996. Fast sorting by reversals. *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science*, 168-185.

[BKS96] M. Blanchette, T. Kunisawa et D. Sankoff. 1996. Parametric genome rearrangement. *Gene*, vol. 172: 11-17.

[BKS99] M. Blanchette, T. Kunisawa et D. Sankoff. 1999. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, vol. 49: 193-203.

[BMY01] D.A. Bader, B.M.E. Moret et M. Yan. 2001. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, vol. 8: 483-491.

[BP96] V. Bafna et P.A. Pevzner. 1996. Genome rearrangements and sorting by reversals. *SIAM Journal of Computing*, vol. 25: 272-289.

[BP98] V. Bafna et P.A. Pevzner. 1998. Sorting by transpositions. *Siam Journal on Discrete Mathematics*, vol. 11: 224-240.

[BP02] G. Bourque et P.A. Pevzner. 2002. Genome-scale evolution: Reconstructing gene orders in the ancestral species. *Genome Research*, vol. 12: 26-36.

[CA99a] A. Caprara. 1999. On the tightness of the alternating-cycle lower bound for sorting by reversals. *Journal of Combinatorial Optimization*, vol. 3: 149-182.

[CH98] D.A. Christie. 1998. A 3/2-approximation algorithm for sorting by reversals. *Proceedings of the 9th annual ACM-SIAM symposium on discrete algorithms*, 244-252.

[CI01] D.A. Christie et R.W. Irving. 2001. Sorting strings by reversals and by transpositions. *Siam Journal on Discrete Mathematics*, vol. 14: 193-206.

[CLN01] A. Caprara, G. Lancia et S.K Ng. 2001. Sorting permutations by reversals through branch-and-price. *Journal on Computing*, vol. 13: 224-244.

[CR02] A. Caprara et R. Rizzi. 2002. Improved approximation for breakpoint graph decomposition and sorting by reversals. *Journal of Combinatorial Optimization*, vol. 6: 157-182.

[DEE02] D.A. Dalevi, N. Eriksen, K. Eriksson et S.G.E. Andersson. 2002. Measuring genome divergence in bacteria: A case study using *Chlamydia* data. *Journal of Molecular Evolution*, vol. 55: 24-36.

[EM00] N. El-Mabrouk. 2000. Sorting signed permutations by reversals and insertions / deletions of contiguous segments. *Journal of Discrete Algorithms*, vol. 1(1): 105-122.

[EM02] N. El-Mabrouk. 2002. Reconstructing an ancestral genome using minimum segments duplication and reversals. *Journal of Computer and Systems Sciences*, vol. 65: 442-464.

[GN02] J. Gramm et R. Niedermeier. 2002. Breakpoint medians and breakpoint phylogenies: A fixed-parameter approach. *Bioinformatics*, vol. 18: S128-S139.

[HA03] T. Hartman. 2003. A simpler 1,5-approximation algorithm for sorting by transpositions. *Proceedings of CPM'03*, à paraître.

[HA96] S. Hannenhalli. 1996. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, vol. 71: 137-151.

[HP95a] S. Hannenhalli et P.A. Pevzner. 1995. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Proceedings of the 27th ACM-SIAM Symposium on the Theory of Computing*, 178-189.
Aussi dans *Journal of the ACM* (1999) vol. 46(1): 1-27.

[HP95b] S. Hannenhalli et P.A. Pevzner. 1995. Transforming men into mice: Polynomial algorithm for genomic distance problem. *Proceedings of the IEEE 36th Annual Symposium on Foundations of Computer Science*, 581-592. IEEE Computer Society, Los Alamitos, CA.

[HV98] L.S. Heath et J.P.C. Vergara. 1998. Sorting by bounded block moves. *Discrete Applied Mathematics*, vol. 88: 181-206.

[HV00] L.S. Heath et J.P.C. Vergara. 2000. Sorting by short block moves. *Algorithmica*, vol. 28: 323-352.

[KS95] J. Kececioglu et D. Sankoff. 1995. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, vol. 13: 180-210.

[KST97] H. Kaplan, R. Shamir et R.E. Tarjan. 1997. A faster and simpler algorithm for sorting signed permutations by reversals. *Procedures of the 8th annual ACM-SIAM Symposium on Discrete Algorithms (OSODA 97*, pages 344-351.
Aussi dans *SIAM Journal on Computing* (2000) vol. 29(3) : 880-892.

[KU01] T. Kunisawa. 2001. Gene arrangements and phylogeny in the class proteobacteria. *Journal of Theoretical Biology*, vol. 213: 9-19.

[LET03] J.-F. Lefebvre, N. El-Mabrouk, E. Tillier et D. Sankoff. Detection and validation of single gene inversions. *Bioinformatics*, à paraître.

[MBW02] B.M.E. Moret, D.A. Bader et T. Warnow. 2002. High performance algorithm engineering for computational phylogenetics. *Journal of Supercomputing*, vol. 22: 99-110.

[PH88] J.D. Palmer et L.A. Herbon. 1988. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, vol. 27: 87-97.

[PS98] I. Pe'er et R. Shamir. 1998. The median problems for breakpoints are NP-complete. Manuscripte, Université de Washington.

[PT03] P. Pevzner et G. Tesler. 2003. Transforming men into mice: evidence against the Nadeau-Taylor chromosomal breakage model. *Proceedings of the 7th Annual International Conference on Computational Molecular Biology* (RECOMB 2003), à paraître.

[SA99] D. Sankoff. 1999. Genome rearrangement with gene families. *Bioinformatics*, vol. 15: 909-917.

[SA02] D. Sankoff. 2002. Short inversions and conserved gene cluster. *Bioinformatics*, vol. 18: 1305-1308.

[SB97] D. Sankoff et M. Blanchette. 1997. The median problem for breakpoints in comparative genomics. *Lecture Notes in Computer Science*, vol. 1276: 251-263.

[SB99] D. Sankoff et M. Blanchette. 1999. Phylogenetics invariants for genome rearrangements. *Journal of Computational Biology*, vol. 6: 431-445.

[SBD00] D. Sankoff, D. Bryant, M. Deneault, B.F. Lang et G. Burger. 2000. Early eukaryote evolution based on mitochondrial gene order breakpoints. *Journal of Computational Biology*, vol. 7: 521-535.

[SE00] D. Sankoff et N. El-Mabrouk. 2000. Duplication, rearrangement and reconciliation. Comparative genomics: empirical and analytical approaches to gene order dynamics, map alignment and the evolution of gene families (DCAF), vol. 1: 537-550.

[SI02] A.C. Siepel. 2002. An algorithm to find all sorting reversals. *RECOMB 2002*, 281-290.

[TC00] E.R.M. Tillier et R.A. Collins. 2000. Genome rearrangement by replication-directed translocation. *Nature Genetics*, vol. 26: 195-197.

[YCD00] I.Yanai, C.J. Camacho et C. DeLisi. 2000. Predictions of gene family distributions in microbial genomes: evolution by gene duplication and modification. *The American Physical Society*, vol. 85: 2641-2644.

[WEH82] W.A. Watterson, W.J. Ewens, T.E. Hall et A. Morgan. 1982. The chromosome inversion problem. *Journal of Theoretical Biology*, vol. 99: 1-7.

[WG02] S. Wu et X. Gu. 2002. Multiple genome rearrangement by reversals. *Pacific Symposium on Biocomputing 2002*, 259-270.

[WW01] L.S. Wang et T. Warnow. 2001. Estimating true evolutionary distances between genomes. *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, 637-646.

Annexe A :

Code en langage C de la fonction de l'algorithme INV permettant de trouver toutes les solutions.

```
/*---------------------------- FONCTION TROUVER_LISTE ----------------------
 * Fonction permettant de trouver toutes les solutions possibles après chaque
 * inversion.
*-------------------------------------------------------------------------------------*/

inverse_elt* trouver_liste(int* perm_inverse, int *perm_source, float *perm_distance,
                           int borne_inf, int borne_sup)  {

  int i, j, nb_safe, intervalle_1, intervalle_2, fin = 0, debut, element_safe;
  int p0, p1, p2, p3, compteur=0, compteur1=0, arret=0, cnt_elt, compteur_liste1;
  int x, z, y, elt_existe, elt, dist1, dist2, d1, d2, var_d, compteur_liste2, decompte1;
  int tab_choix[MAX_SIZE];
  cycle_cc *ptr_cycle, *ptr_inv1, *ptr_inv2, *p_temp, *p_temp1, *p_temp2, *p_tmp2;
  inverse_elt *ptr_liste, *p_courant, *prec, *p_interv, *p_total, *p_total_deb, *p_temp_tot;
  int del_der = size/2, limite, minimum = 0, tout, moitie, quart, trois_quart ;
  float var_axe;

  srand((unsigned)time(NULL));
  limite = 1000;  // limite d'inversions gardees en memoire.

  p_total = (inverse_elt *) malloc(sizeof(inverse_elt));
  p_total_deb = p_total;
  p_total->elt_safe = -9;

  for (i=0 ; (i<nb_c); i++) {
    compteur_liste1 = compteur_liste2 = 0;
    tab_cycles[i]     = 0;
    cycle_inv[2*i]    = (cycle_cc *) malloc(sizeof(cycle_cc));
    cycle_inv[(2*i)+1] = (cycle_cc *) malloc(sizeof(cycle_cc));
    ptr_inv1          = cycle_inv[2*i];
    ptr_inv2          = cycle_inv[(2*i)+1];
    ptr_cycle         = cycle_nb[i];

    if( (perm_inverse[2 * ptr_cycle->elt] % 2) == 0 )
      ptr_inv1->elt = perm_inverse[2 * ptr_cycle->elt];     //position du premier elt pair
    else
```

```
 ptr_inv1->elt = perm_inverse[2 * ptr_cycle->elt] - 1; //position du premier elt impair

ptr_inv2->suivant = NULL;
ptr_inv1->suivant = NULL;

// si le cycle est compris entre les bornes
if(ptr_inv1->elt >= borne_inf && ptr_inv1->elt <= borne_sup) {
  fin = ptr_inv1->elt;          // fin du cycle.
  p0 = fin;                     // position du premier elt.
  p1 = p0 + 1;                  // position de son arc noir.
  p2 = perm_source[p1];         // elt de son arc noir.
  p3 = perm_inverse[MODULO(p2)]; // position de l'arc gris de p2
  compteur_liste1 ++;

  while (p3 != fin){
       p_temp = (cycle_cc *) malloc(sizeof(cycle_cc)); // allocation
       if ( (p3 % 2) == 0 ) {       // on ajoute l'elt dans la bonne liste
         ptr_inv1->suivant = p_temp;
         ptr_inv1 = p_temp;
         ptr_inv1->elt = p3;
         ptr_inv1->suivant = NULL;
         compteur_liste1++;
       }
       else {
         ptr_inv2->suivant = p_temp;
         ptr_inv2 = p_temp;
         ptr_inv2->elt = p3 - 1;
         ptr_inv2->suivant = NULL;
         compteur_liste2++;
       }
       p0 = p3;
       p1 = MODULO(p0);
       p2 = perm_source[p1];
       p3 = perm_inverse[MODULO(p2)];
  } // fin while

  if(compteur_liste1 * compteur_liste2 != 0) {
       ptr_inv1 = cycle_inv[2*i];
       while(ptr_inv1 != NULL) {
         ptr_inv2 = cycle_inv[(2*i)+1];
         ptr_inv2 = ptr_inv2->suivant;
         while(ptr_inv2 != NULL && compteur < limite) {
           if(p_total->elt_safe != -9) {
             p_temp_tot = (inverse_elt *) malloc(sizeof(inverse_elt));
             p_total->next = p_temp_tot;
             p_total = p_total->next;
           }
           p_total->elt_safe = ptr_inv1->elt; // pointeur vers toutes les inversions possibles.
           p_total->nbre_elts = ptr_inv2->elt;
```

```
            p_total->next = NULL;
            ptr_inv2 = ptr_inv2->suivant;
            compteur++;
           }
           ptr_inv1 = ptr_inv1->suivant;
          }
    }
    ptr_inv1 = NULL;
    ptr_inv2 = NULL;

  }
}

for(i=0; i<limite; i++)
  tab_choix[i]=-1;

ptr_liste = (inverse_elt *) malloc(sizeof(inverse_elt));
if(ptr_liste == NULL) { printf("pb alloc ptr_liste \n") ; exit(1); }

ptr_liste->elt_safe = -9;
p_courant = ptr_liste;  //p_courant pointe sur le debut de la liste

for(i = 0; i<limite && i<compteur; i++) {
  do{
    nb_safe = 1;
    j = rand()%compteur;
    for(z=0; z<limite; z++)
         if(tab_choix[z] == j)
           nb_safe = 0;
  }while(nb_safe == 0);
  tab_choix[i] = j;

  p_total = p_total_deb; // repointe au debut des choix
  cnt_elt = 0;

  while( cnt_elt < j ) {
    p_total = p_total->next;
    cnt_elt++;
  }

  if( p_total->elt_safe < p_total->nbre_elts) {
    intervalle_1 = p_total->elt_safe;
    intervalle_2 = p_total->nbre_elts;
  }
  else {
    intervalle_2 = p_total->elt_safe;
    intervalle_1 = p_total->nbre_elts;
  }
```

```
if(pourcent == 1) {
 d1 = perm_distance[(intervalle_1+1) / 2];    // le debut du premier gene
 d2 = perm_distance[(intervalle_2+1) / 2];    // la fin du deuxieme gene = le debut du
                                                 gene d'apres
 tout = 100.0;                                // dans le cas des distances en %
}
else {
 d1 = (intervalle_1 / 2) + (int)perm_distance[(intervalle_1 / 2) + 1];
 d2 = (intervalle_2 / 2) + (int)perm_distance[(intervalle_2 / 2)];
 tout = (size/2 + (int)perm_distance[0]);  // on considere TOUS les genes
}

moitie = tout/2;
quart = moitie/2;
trois_quart = 3*tout/4;
if((d1 < moitie)&&(d2 > moitie)) {
 if(d1 <= quart && d2 >= trois_quart){
      dist1 = d1;
      dist2 = tout - d2;
 }
 else
      if(d1 <= quart || d2 >= trois_quart){
       if((d1 + tout - d2) <= (d2 - d1)){
        dist1 = d1;
        dist2 = tout - d2;
       }
       else{
        dist1 = moitie - d1;
        dist2 = d2 - moitie;
       }
      }
      else { /* considerer les quartiles 2 et 3 */
       dist1 = moitie - d1;
       dist2 = d2 - moitie;
      }
 var_axe = (float)(abs(dist1-dist2))/(float)(dist1+dist2);
}
else
 var_axe = 1.0;

if(pourcent == 1) {
 if( (d2 - d1) > 50 )                         // dans le cas des pourcentages
      var_d = 100 - d2 + d1;
 else
      var_d = d2 - d1;
}
else {
 if( (d2 - d1) > ((size/2 + (int)perm_distance[0])/2) )  // dans le cas des genes
      var_d = (size/2) + (int)perm_distance[0] - d2 + d1;
```

```
  else
       var_d = d2 - d1;
 }

 if( ((intervalle_2 % 2) != 0) && (intervalle_2 > (intervalle_1 + 1)) )
  intervalle_2 = intervalle_2 - 1;

 if(intervalle_1 != intervalle_2) {
  if(ptr_liste->elt_safe == -9) {
       ptr_liste->elt_safe  = intervalle_1;       //element_safe;
       ptr_liste->nbre_elts = intervalle_2;       //(intervalle_2 - intervalle_1) + 1;
       ptr_liste->type      = 1;
       ptr_liste->dist      = var_d;
       ptr_liste->cnt_axe   = var_axe;
       ptr_liste->next      = NULL;
  }
  else {  //chercher ou placer l'element
       ptr_liste = p_courant;
       prec = ptr_liste;
       //creer une case
       p_interv = (inverse_elt *) malloc(sizeof(inverse_elt));
       if(p_interv == NULL)  { printf("pb alloc p_interv \n"); exit(1); }
       p_interv->elt_safe  = intervalle_1;
       p_interv->nbre_elts = intervalle_2;
       p_interv->type      = 1;
       p_interv->dist      = var_d;
       p_interv->cnt_axe   = var_axe;
       p_interv->next      = NULL;
       while(ptr_liste != NULL && fin != 1){
        if(ptr_liste->next == NULL) {
          ptr_liste->next = p_interv;
          ptr_liste = p_interv;
        }
        prec = ptr_liste;
        ptr_liste = ptr_liste->next;
       }
  }
 }
 else
  i--;

 ptr_liste = p_courant;
 fin = 0;
}
return p_courant;
}// FIN trouver_liste
```

Annexe B :

Code en langage C de la fonction « Construire-G^C » de l'algorithme INV.

```
/*------------------------- FONCTION CONSTRUIRE_G^C ----------------------------
 * Fonction qui ajoute les éléments de A_G dans la permutation Ĝ^C pour obtenir G^C
 *
 *-------------------------------------------------------------------------------------------*/
void construire_Gc(int* source, int* perm_del, int* perm_ins, int taille, int size_reduit) {
 int i, j, taille_r, k = 0, indirect = 0, rendu = 0, gris, debut, noir, verifie, premier, deuxieme;
 int perm_inverse[MAX_SIZE], remplace[MAX_SIZE];
 int tab_cycle[MAX_SIZE], tab_visite[MAX_SIZE];

 // construction de la permutation telle que x -> (2x-1, 2x)
 remplace[0] = 0;
 j=1;
 for(i=0; i<taille; i++){
  if(source[i] > 0) {
   remplace[j] = source[i]*2 - 1;
   remplace[j+1] = source[i]*2;
  }
  else {
   remplace[j] = abs(source[i]) * 2;
   remplace[j+1] = abs(source[i]) * 2 - 1;
  }
  j+=2;
 }
 taille_r = j;
 remplace[taille_r] = taille_r;

 //construction de la permutation inverse de remplace
 for(i=0; i<=taille_r; i++)
  perm_inverse[remplace[i]] = i;

 for(i=0; i<=taille_r; i++) {
  tab_cycle[i] = -1; // tableau indiquant si l'arc i fait parti d'un cycle direct (0) ou indirect (1).
  tab_visite[i] = -1; // tableau indiquant que le ieme arc visite est tab_visite[i].
 }
 j=0;
```

```
//Parcours des arcs pour optimiser l'insertion des éléments à effacer.
for(i=0; i<taille_r; i+=2) {
 indirect = 0;                    // on n'est pas encore passe par cet arc.
 if(tab_cycle[i] == -1) {         // sinon, on n'a pas le choix d'insertion
    debut = i;                    // position debut
    gris  = perm_inverse[MODULO(remplace[debut])];  // position arc gris
    noir  = MODULO(gris);                           // position arc noir.
    tab_visite[k++] = debut;
    tab_visite[k++] = gris;

    if(gris%2 == 0)   verifie = gris;
    else              verifie = noir;
    if( ((remplace[verifie]+1)/2 <= size_reduit) &&
        ((remplace[verifie+2]+1)/2 > size_reduit) )
         indirect = 1;   // indique que l'arc est indirect.
    while(noir != i){
     debut = noir;
     gris  = perm_inverse[MODULO(remplace[debut])];
     noir  = MODULO(gris);
     tab_visite[k++] = debut;
     tab_visite[k++]  = gris;
     if(gris%2 == 0)   verifie = gris;
     else              verifie = noir;
     if( ((remplace[verifie]+1)/2 < size_reduit ) &&
         (perm_ins[(remplace[verifie]+1)/2] != 0) )
       indirect = 1;   // indique que l'arc est indirect.
    }
   }
   for(j=rendu; j<k; j++)    tab_cycle[tab_visite[j]] = indirect;
   rendu = k;
 }
 // table_cycle est construite
 for(i=0; i<taille_r; i+=2) // parcours des arcs  si Xa existe et X appartient aux communs
  if( (((remplace[i]+1)/2) <= size_reduit) && (perm_del[(remplace[i]+1)/2] != 0) ) {
    if( ((remplace[i+2]+1)/2) > size_reduit ) {
       premier = i;
       if (tab_cycle[premier] != 1) {
         do{
          i+=2;
         }while( (remplace[i]+1)/2 >= size_reduit && (i!=taille_r-1));
         deuxieme = i;
         if(tab_cycle[deuxieme] == 1) {
          perm_del[remplace[(deuxieme+1)/2]] = perm_del[remplace[(premier+1)/2]];
          perm_del[remplace[(premier+1)/2]] = 0;
         }
       }
    }
  }
}//---- FIN construire_Gc
```

Annexe C :

Code en langage C de la librairie « Inversion_genome.h » utilisée dans l'algorithme INV.

```
#ifndef INVERSION_GENOME
#include<time.h>

#define INVERSION_GENOME
#define MAX_SIZE 14000                          // Nombre maximum de gènes
#define MIN(x,y) ((x)>(y)?(y):(x))
#define MAX(x,y) ((x)<(y)?(y):(x))
#define MODULO(x) (((x % 2) == 0)?(x+1):(x-1))
#define VAL_ABS(x,y) ((x<y)?(y-x):(x-y))
#define ARC_DEBUT -1
#define ARC_FIN -2
#define ORIENTE -9
#define NON_ORIENTE -8
#define HURDLE -3
#define SUPER_HURDLE -4
#define NON_HURDLE -5
#define CYCLE_T1 -10  //cycles de taille1
#define nb_bit 3            //nb de bit par entier
#define marge 0.10

typedef struct cc{        //cette structure est utilisee pour garder les arcs qui forment les cycles
  int elt;
  struct cc *suivant;
} cycle_cc;
cycle_cc *cycle_nb[MAX_SIZE], *ptr;
cycle_cc *cycle_inv[MAX_SIZE]; //tableau contenant les arcs noirs des cycles pouvant etre
inversés.

typedef struct composantes{
  int elt_D;   //position debut de la composante
  int elt_F;   //position fin de la composante
  int elt_P;    //parite
  int elt_H;   //super, non, hurdles
  struct composantes *next;
}cycle_comp;
cycle_comp *p_crt, *p_tmp;
```

```
typedef struct cycles{
  int elt_D;  //position debut
  int elt_F;   //position fin   de chacun des arcs qui forment le cycle
  int debut;  //position debut du cycle
  int fin;      //position fin
  int elt_P;   //parite
  int elt_H;  //super, non, hurdles
  int nb_del;  //nombre de segments a effacer dans le cycle.
  struct cycles *next;
}cycle_cc1;
cycle_cc1 *p_debut, *ptr1, *p_liste, *p_cut, *p_cut1;

typedef struct inversion1{
  int type;          // 0 -> mauvaise compo , 1 -> bonne compo .
  int elt_safe;     //position debut
  int nbre_elts;   //position fin   de chacun des arcs qui forment le cycle
  int reduit;        // 0 si ca reduit pas le nb de segment a enlever a la fin, 1 si oui.
  int type_inv;    // 0 vers la gauche, 1 vers la droite.
  int dist;          // Taille de l'inversion.
  float cnt_axe;  // Rapport entre la position de l'inversion et l'axe de réplication.
  struct inversion1 *next;
}inverse_elt;
inverse_elt *debut_hdl, *debut_liste, *debut_meilleur;

int vecteurs_safe[MAX_SIZE], source[MAX_SIZE];

//tableau contenant les positions debut et fin des cycles et composantes :
int debut_fin_cycle[MAX_SIZE], debut_fin_compos[MAX_SIZE];

//contient les cycles qui forment les composantes connexes et la parite de chaque composante
int composante_cx[MAX_SIZE], parite[MAX_SIZE], cycle[MAX_SIZE];

int overlap[MAX_SIZE][MAX_SIZE], score[MAX_SIZE];
int debut[MAX_SIZE], arcGris[MAX_SIZE];

int perm_del_global[MAX_SIZE], tab_cycles[MAX_SIZE];
float perm_distance[MAX_SIZE];

int size, pos_debut, pos_fin, arcDF;
int nbre_cycles, nbre_compos, nb_c, nombre_composantes, a_afficher;
int nb_super_hurdle, nb_hurdles;
int distance, dist_inv, compt_mvt, compo_nombre;
int inversion_finie, compt_mvt, dist, nb_reduction_best;
int arcDF, bonne_comp, fois_1, free_elt, best, best_dist, compo_resolue;
extern int pos_debut, pos_fin, arcDF;
int axe_repl;
int etape2, size_tot_H, size_plus_G, borne_inf, borne_sup, limite_atteinte;
#endif
```