Université de Montréal

Évolution des génomes par mutations locales et globales - Une approche d'alignement

par
Billel Benzaid

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Juin, 2016

© Billel Benzaid, 2016.

RÉSUMÉ

Durant leur évolution, les génomes accumulent des mutations pouvant affecter d'un nucléotide à plusieurs gènes. Les modifications au niveau du nombre et de l'organisation des gènes dans les génomes sont dues à des mutations globales, telles que les duplications, les pertes et les réarrangements. En comparant les ordres de gènes des génomes, il est possible d'inférer les événements évolutifs les plus fréquents, le contenu en gènes des espèces ancestrales ainsi que les histoires évolutives ayant menées aux ordres observés.

Dans cette thèse, nous nous intéressons au développement de nouvelles méthodes algorithmiques, par approche d'alignement, afin d'analyser ces différents aspects de l'évolution des génomes. Nous nous intéressons à la comparaison de deux ou d'un ensemble de génomes reliés par une phylogénie, en tenant compte des mutations globales.

Pour commencer, nous étudions la complexité théorique de plusieurs variantes du problème de l'alignement de deux ordres de gènes par duplications et pertes, ainsi que de l'approximabilité de ces problèmes. Nous rappelons ensuite les algorithmes exacts, en temps exponentiel, existants, et développons des heuristiques efficaces.

Nous proposons, dans un premier temps, DLAlign, une heuristique quadratique pour le problème d'alignement de deux ordres de gènes par duplications et pertes. Ensuite, nous présentons, OrthoAlign, une extension de DLAlign, qui considère, en plus des duplications et pertes, les réarrangements et les substitutions.

Nous abordons également le problème de l'alignement phylogénétique de génomes. Pour commencer, l'heuristique OrthoAlign est adaptée afin de permettre l'inférence de génomes ancestraux au noeuds internes d'un arbre phylogénétique.

Nous proposons enfin, MultiOrthoAlign, une heuristique plus robuste, basée sur la médiane, pour l'inférence de génomes ancestraux et d'histoires évolutives d'un ensemble

de génomes représentés aux feuilles d'un arbre phylogénétique.

Mots clés : Algorithme, alignement, ordres de gènes, programmation dynamique, complexité, duplication, perte, réarrangements génomiques

During the evolution process, genomes accumulate mutations that may affect the genome at different levels, ranging from one base to the overall gene content. Global mutations affecting gene content and organization are mainly duplications, losses and rearrangements. By comparing gene orders, it is possible to infer the most frequent events, the gene content in the ancestral genomes and the evolutionary histories of the observed gene orders.

In this thesis, we are interested in developing new algorithmic methods based on an alignment approach for comparing two or a set of genomes represented as gene orders and related through a phylogenetic tree, based on global mutations.

We study the theoretical complexity and the approximability of different variants of the two gene orders alignment problem by duplications and losses. Then, we present the existing exact exponential time algorithms, and develop efficient heuristics for these problems.

First, we developed DLAlign, a quadratic time heuristic for the two gene orders alignment problem by duplications and losses. Then, we developed OrthoAlign, a generalization of DLAlign, accounting for most genome-wide evolutionary events such as duplications, losses, rearrangements and substitutions.

We also study the phylogenetic alignment problem. First, we adapt our heuristic OrthoAlign in order to infer ancestral genomes at the internal nodes of a given phylogenetic tree.

Finally, we developed MultiOrthoAlign, a more robust heuristic, based on the median problem, for the inference of ancestral genomes and evolutionary histories of extent genomes labeling leaves of a phylogenetic tree.

Key words : Algorithm, alignment, gene orders, dynamic programming, complexity, duplication, loss, genomic rearrangements

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

ADN	Acide DésoxyriboNucléique
ARN	Acide RiboNucléique
ARNt	Acide RiboNucléique de transfert
DCJ	Double Cut and Join
LCB	Locally collinear blocks
LIS	Longest Increasing Subsequence
MEM	Maximal Exact Matches
MUM	Maximal Unique Matches
NPO	NP-Optimization
PTAS	Polynomial Time Approximation Scheme
SCJ	Single Cut or Join
SCJ2	Single Cut and Join

À mes parents, mes frères, mes soeurs, mes nièces et mon neveu.

Voyou ! un jour, ISA, tu liras cette thèse.

REMERCIEMENTS

Je tiens tout d'abord à remercier vivement ma directrice de recherche, Nadia El-Mabrouk, pour sa grande disponibilité, son aide précieuse dans la rédaction des articles, ses conseils constructifs dans la préparation des présentations orales, la rédaction et la relecture de cette thèse. Merci Nadia de m'avoir accueilli dans ton laboratoire.

Je remercie également mon co-directeur, Miklós Csurös, pour ses conseils précieux me permettant de réaliser cette thèse.

Mes remerciements vont ensuite à mes collègues avec qui j'ai eu le plaisir de travailler dans le laboratoire de Nadia : Krister Swenson, Patrick Holloway, Olivier Tremblay Savard, Manuel Lafond, Yves Gagnon, Emmanuel Noutahi et Virginie Calderon. Je remercie particulièrement Krister pour ses conseils, ses discussions qui m'ont aidé à la réalisation d'un de mes projets.

Je remercie l'ensemble des membres du jury pour avoir accepté d'évaluer cette thèse.

Mes remerciements vont particulièrement à mes parents qui m'ont constamment encouragé et soutenu moralement, de loin, tout au long de ces années. Je remercie également les autres membres de ma famille (frères, soeurs, oncles, tante, cousins et cousines, etc.) pour m'avoir soutenu, de loin, durant mes études doctorales

Enfin, je remercie les organismes qui m'ont supporté financièrement durant mes études doctorales : le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG), les bourses d'excellence du département DIRO.

INTRODUCTION

Grâce aux progrès réalisés dans les technologies de séquençage, les génomes complets de plus d'un millier d'espèces sont actuellement disponibles dans les banques de données publiques[1]. Cela rend possible l'étude de l'évolution des espèces à partir de leur matériel génétique complet, plutôt qu'à partir de gènes individuels. Le fait de disposer des génomes complets permet d'étudier, non seulement les mutations locales affectant les séquences de nucléotides, mais également les mutations globales affectant l'organisation en gènes des génomes. Pour ce faire, de nouvelles méthodes algorithmiques permettant de comparer des génomes complets en se basant sur les opérations (ou événements) affectant les nucléotides, mais également les opérations de duplication et perte affectant le contenu en gènes, et les opérations de réarrangement affectant l'ordre des gènes dans les génomes, doivent être mises au point.

L'objectif de cette thèse est de développer de telles méthodes algorithmiques, les plus précises et efficaces possibles, permettant de comparer des génomes représentés sous forme d'ordres de gènes. Les événements évolutifs considérés sont les duplications de gènes, les pertes de gènes, les réarrangements (inversions et transpositions) et les substitutions de gènes. L'approche algorithmique que l'on considère consiste à explorer les suites d'opérations nécessaires pour transformer un génome en un autre. Pour un ensemble donné d'opérations, la plus courte séquence (ou scénario optimal) de ces opérations est considérée comme la distance entre les génomes. La plupart des problèmes de calcul de distances entre deux génomes sont NP-difficile [64, 66, 71, 262].

Dans cette thèse, nous considérons l'hypothèse d'opérations non-chevauchantes, autrement dit, chaque site (position dans le génome) est affecté par au plus un événement évolutif. Ces événements peuvent donc être observés (ou *visibles*) dans l'alignement des génomes, et par conséquent, le problème de comparaison d'ordres de gènes se ramène

[1] Avril 2014, http ://www.the-scientist.com/ ?articles.view/articleNo/39742/title/Sequencing-the -Tree-of-Life/

au problème d'alignement d'ordres de gènes. À partir d'un tel alignement, il est possible de reconstruire l'histoire évolutive des génomes ainsi que leurs ancêtres. L'hypothèse d'opérations non-chevauchantes a été introduite auparavant par Schoniger et Waterman [222] dans le but de proposer une solution polynomiale au problème d'alignement de séquences de nucléotides par inversions, étant donné que la complexité du problème dans le cas général (i.e. sans aucune restriction pour les opérations) était inconnue. Cette hypothèse a permis de proposer des solutions polynomiales aux problèmes d'alignement de séquences de nucléotides en considérant les opérations d'inversion, de transposition et de duplication en tandem [75, 173, 245]. Dans le cas de comparaison d'ordres de gènes, l'hypothèse d'opérations non-chevauchantes permet d'étudier l'évolution d'espèces évolutivement proches.

La restriction du problème d'alignement de deux ordres de gènes aux événements de duplication et perte a été prouvée NP-difficile [13]. Les deux algorithmes exacts de programmation linéaires [13, 149], développés pour cette restriction, sont exponentiels dans le pire des cas. Pour les applications biologiques, le temps nécessaire pour déterminer une solution est généralement déterminant dans le choix de l'algorithme. Pour des algorithmes permettant d'obtenir des résultats presque similaires, le plus rapide est généralement considéré comme étant le meilleur. C'est pourquoi, dans cette thèse, nous nous intéressons à développer des algorithmes, les plus efficaces possibles au niveau du temps d'exécution, sans perdre en précision.

Sachant que le problème d'alignement de deux ordres de gènes par duplications et pertes est NP-difficile, nous avons soulevé la question de la faisabilité de problèmes sous-jacents. Étant donné un alignement, que l'on peut obtenir par une méthode de programmation dynamique, qu'en est-il du problème de l'étiquetage des "gaps" afin d'obtenir une histoire réalisable expliquant l'évolution des deux génomes alignés à partir d'un ancêtre commun ? D'autre part, étant donné un alignement interprété (étiqueté) non-réalisable (pour des raisons d'explications circulaires des "gaps"), comment corri-

ger minimalement cet alignement pour qu'il le devienne (i.e. réalisable) ? Plus précisément, nous soulevons la question d'association d'événements de duplication et perte à un alignement initial et le problème de désambiguïser un alignement déjà associé à des événements évolutifs.

Cette thèse porte à la fois sur l'étude de la complexité et le développement d'heuristiques efficaces pour la résolution de ces problèmes, et sur le développement d'heuristiques efficaces pour la résolution du problème d'alignement de deux ou plusieurs ordres de gènes. En particulier, trois heuristiques principales ont été proposées. Ces heuristiques sont polynomiales en temps et se basent sur la programmation dynamique.

La première heuristique, DLAlign, concerne le problème d'alignement de deux ordres de gènes par duplications et pertes. Les résultats de l'étude des problèmes sous-jacents nous ont permis de concevoir une heuristique garantissant l'optimalité des solutions dans le cas où les données considérées vérifient certaines propriétés.

Étant donné que des traces d'événements d'inversion, de transposition et de substitution peuvent être observées dans les génomes, ces événements ont alors été intégrés dans le modèle évolutif, conduisant à la conception d'une seconde heuristique plus générale, OrthoAlign. Le problème de l'alignement phylogénétique de génomes a ensuite été abordé. Dans un premier temps, l'heuristique OrthoAlign a été adaptée afin de permettre l'inférence de génomes ancestraux au noeuds internes d'un arbre phylogénétique, et l'analyse de l'évolution de génomes pouvant être sujets à l'ensemble des événements considérés.

Enfin, le problème d'alignement de trois ordres de gènes a été abordé, conduisant à la conception d'une troisième heuristique plus robuste, MultiOrthoAlign, basée sur la médiane, permettant d'étendre l'inférence de génomes ancestraux de cerises (i.e. arbres ayant deux feuilles) à une phylogénie.

La thèse est subdivisée de la façon suivante. Le chapitre 1 introduit les notions de base dans le domaine de la biologie et de la complexité des algorithmes, nécessaires à la

compréhension des chapitres suivants.

Le chapitre 2 constitue un état de l'art de l'alignement de séquences de nucléotides. La première partie de ce chapitre est consacrée au cas particulier d'alignement de courtes séquences. Nous y exposons le problème d'alignement de séquences puis, nous citons brièvement les algorithmes classiques pour sa résolution. La deuxième partie est consacrée au cas général de séquences complètes de génomes. Dans un premier temps, nous présentons la stratégie générale partagée par la plupart des algorithmes d'alignement de génomes complets. Ensuite, nous présentons les algorithmes d'alignement existants les plus connus, qui tiennent compte des réarrangements.

Le chapitre 3 introduit les problématiques liées à l'étude des réarrangements génomiques, de la comparaison de deux ordres de gènes à l'inférence d'ordres ancestraux dans un arbre phylogénétique. Dans le cas de présence de gènes dupliqués dans les génomes, différents modèles de couplages existent afin de désambiguïser ces gènes. Sans entrer dans tous les détails des algorithmes, nous mettons en relief les différents problèmes soulevés, et leurs complexités. En particulier, nous présentons les méthodes les plus connues pour l'inférence d'ordres ancestraux dans un arbre phylogénétique.

Le chapitre 4 présente le problème d'alignement d'ordres de gènes. Nous y exposons le problème général, puis sa restriction aux opérations de duplication et perte. Enfin, nous présentons en détails le premier algorithme exact PBLP [149], développé dans notre laboratoire, pour la résolution de cette restriction.

Le chapitre 5 expose les premiers algorithmes développés dans le cadre de cette thèse. Dans un premier temps, nous étudions la complexité d'une restriction du problème d'alignement de deux ordres de gènes par duplications et pertes. Nous présentons ensuite l'heuristique DLAlign, en temps quadratique, pour le problème d'alignement de deux ordres de gènes par duplications et pertes. Les résultats expérimentaux ont démontré la qualité de DLAlign en terme de solutions fournies, et son efficacité en terme de temps d'exécution. Les travaux présentés dans ce chapitre ont fait l'objet de deux articles. Le

premier a été présenté lors de la conférence LATA 2013 [30], et le deuxième, version plus complète du premier, a été soumis au journal *IEEE/ACM Transactions on Computational Biology and Bioinformatics TCBB* [31].

Le chapitre 6 est consacré à la présentation de l'heuristique OrthoAlign, une extension de DLAlign aux événements de réarrangement et de substitution. L'intégration des inversions et l'utilisation d'un coût quelconque pour les différents événements entraînent cependant l'accroissement de la complexité théorique de l'algorithme, de quadratique à cubique. Les résultats obtenus sur des jeux de données simulés montrent que notre heuristique a une bonne précision en ce qui a trait à l'inférence du nombre et de la distribution des événements dans un arbre phylogénétique. Notre heuristique a également permis d'étudier l'évolution des gènes d'ARN de transfert chez 50 souches de *Bacillus*. Les travaux présentés dans ce chapitre ont fait l'objet d'un article publié dans le journal *Molecular Biology and Evolution* [243].

Le chapitre 7 correspond à un article publié dans le journal *BMC Genomic* [32]. Dans le cadre de ce travail, nous avons présenté l'heuristique multiOrthoAlign, pour l'inférence d'ordres ancestraux dans un arbre phylogénétique donné. Nous avons d'abord généralisé l'heuristique OrthoAlign à l'alignement de trois ordres de gènes afin de pouvoir inférer leur génome médian. Nous avons ensuite utilisé l'heuristique pour l'inférence d'ordres ancestraux dans un arbre phylogénétique donné. Les résultats obtenus sur des jeux de données simulés et biologiques ont montré l'efficacité de notre heuristique en terme de résultats fournis et de temps de calcul, par rapport à l'algorithme exact DupLoCut [13] de complexité exponentielle, qui est le seul algorithme existant pour l'inférence d'ordres ancestraux dans un arbre phylogénétique à partir de l'alignement d'ordres de gènes.

Finalement, nous concluons cette thèse en donnant quelques perspectives de recherche qui émanent des problématiques soulevées par notre travail.

CHAPITRE 1

NOTIONS DE BIOLOGIE ET PRÉ-REQUIS

1.1 Introduction

Ce premier chapitre introduit les principales notions nécessaires à la compréhension des travaux réalisés dans cette thèse. Il s'agit de définitions et résultats standard, s'inscrivant dans le cadre de la biologie et de la complexité des algorithmes, présentés volontairement de manière succincte.

Dans la section 1.2, nous rappelons certaines notions de biologie moléculaire et d'évolution. Dans la section 1.3, nous revenons brièvement sur quelques notions de complexité des algorithmes.

1.2 Notions de biologie

1.2.1 Concepts de biologie moléculaire

Le génome est l'ensemble du matériel génétique d'un organisme. Il est à la base de l'hérédité, le phénomène de transfert d'information d'une cellule mère aux cellules filles. L'ADN (Acide DésoxyriboNucléique) est la macromolécule constituant le génome. L'ADN est constitué d'une suite de quatre nucléotides répartis sur deux brins arrangés en une structure hélicoïdale (voir Figure 1.1). Chaque nucléotide est composé d'une base azotée (Adénine A, Cytosine C, Guanine G, ou Thymine T), d'un sucre (désoxyribose) et d'un groupement phosphate.

Les deux brins de l'ADN sont complémentaires au niveau de leurs bases, une Adénine étant associée à une Thymine, et une Guanine à une Cytosine. Ces deux brins sont orientés en sens opposé, dans le sens 5' vers 3', où le coté 5' (resp. 3') représente l'extrémité de la séquence du brin terminée par un phosphate (resp. un désoxyribose).

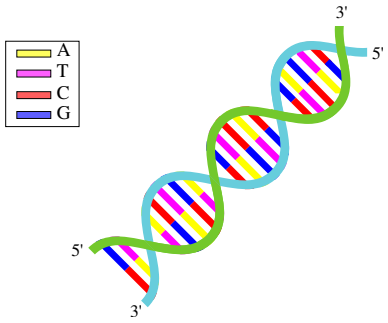


Figure 1.1 : ADN en double brins.

Dans les cellules, L'ADN est organisé en structures appelées *chromosomes*. Les chromosomes sont habituellement présents par paires. L'ensemble des chromosomes d'un organisme vivant constitue son génome. Les génomes peuvent posséder un seul ou plusieurs chromosomes. On les qualifie respectivement d'*unichromosomiques* et de *multichromosomiques*.

Dans les cellules eucaryotes (organismes qui se caractérisent par la présence d'un noyau dans leurs cellules), l'ADN est contenu principalement dans le noyau, alors que dans les cellules procaryotes (organismes dont la structure cellulaire ne comporte pas de noyau), l'ADN est contenu dans le cytoplasme. De plus, les génomes procaryotes sont généralement constitués d'un seul chromosome circulaire. Les génomes eucaryotes, quant-à-eux, sont généralement constitués d'un ensemble de chromosomes linéaires.

Afin de permettre à l'information génétique de se transmettre d'une cellule à d'autres lors de la division cellulaire, l'ADN doit se répliquer. Les deux brins se séparent et chacun sert de modèle pour la formation d'un nouveau brin complémentaire sous l'action d'une enzyme appelée *l'ADN polymérase*. La cellule se retrouve avec deux copies identiques de la molécule d'ADN qui se répartissent entre les deux cellules filles en fin de la division. La taille du génome se mesure en nombre de bases (bases des nucléotides). Généralement, on parle de paires de bases (*pb*) puisque la majorité des génomes étant constituée de doubles brins d'ADN. On emploie souvent les multiples kilo-bases (*kb*)

pour milliers de bases, ou méga-bases (*mb*) pour millions de bases. La taille du génome peut varier de quelques kilo-bases chez les virus, à plusieurs centaines de milliers de méga-bases chez certains eucaryotes.

À une échelle plus grande, un génome peut être vu comme un ensemble de gènes. Un gène est défini comme une portion codante de l'ADN. Il est transcrit en Acide Ribo-Nucléique (ARN) par un processus appelé **transcription**.

La molécule d'ARN est très proche chimiquement de l'ADN avec, cependant, quelques différences importantes. Les molécules d'ARN trouvées dans les cellules sont beaucoup plus courtes que l'ADN du génome, elles ont pour composant un sucre ribose à la place du désoxyribose. La thymine (T) de l'ADN est remplacée par l'Uracile (U) qui s'apparie avec l'Adénine (A). L'ARN se trouve généralement dans les cellules sous forme d'un seul brin.

Certains ARN, que l'on regroupe sous le terme d'ARN-codants (ou ARN messagers) sont traduits en protéines (voir Figure 1.2). Les protéines sont des molécules constituées d'une chaîne polypeptidique d'acides aminés. Elles sont indispensables à la structuration et au fonctionnement des cellules. Les protéines remplissent de nombreux rôles dans la cellule. On peut souligner : un rôle structurel, un rôle de transport de substance dans le sang, un rôle catalytique et un rôle de signalisation.

Durant le processus de traduction de l'ARN en protéines, les nucléotides sont traduits, par triplets (appelés codons), en acides aminés spécifiques suivant le code génétique illustré dans la figure 1.3[1]. Le code génétique comporte 64 codons différents, alors que les acides aminés sont au nombre de 22. Cette correspondance de 64 triplets ou codons avec les 22 acides aminés principaux implique que ce code est très redondant. Nous disons que le code est **dégénéré**.

Pour résumer, le génome est composé de gènes destinés à être transcrits en ARN et ensuite, traduits en protéines permettant de remplir les fonctions nécessaires à la vie

[1] La figure est extraite du site web : http ://ressources.unisciel.fr/biocell/chap6/co/module_Chap6_4.html

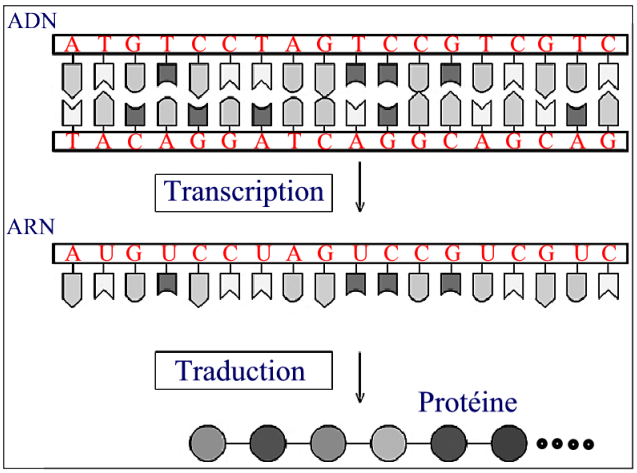


Figure 1.2 : Transformation de l'information génétique en protéine. La transcription d'un gène, situé dans un des deux brins de l'ADN, en ARN puis, la traduction de l'ARN en protéine.

	U		C		A		G		
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC	Phe	UCC	Ser	UAC	Tyr	UGC	Cys	C
	UUA	Leu	UCA	Ser	UAA	Stop	UGA	Stop	A
	UUG	Leu	UCG	Ser	UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC	Leu	CCC	Pro	CAC	His	CGC	Arg	C
	CUA	Leu	CCA	Pro	CAA	Gln	CGA	Arg	A
	CUG	Leu	CCG	Pro	CAG	Gln	CGG	Arg	G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC	Ile	ACC	Thr	AAC	Asn	AGC	Ser	C
	AUA	Ile	ACA	Thr	AAA	Lys	AGA	Arg	A
	AUG	Met	ACG	Thr	AAG	Lys	AGG	Arg	G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC	Val	GCC	Ala	GAC	Asp	GGC	Gly	C
	GUA	Val	GCA	Ala	GAA	Glu	GGA	Gly	A
	GUG	Val	GCG	Ala	GAG	Glu	GGG	Gly	G

Figure 1.3 : Le code génétique. Le codon **AUG** code pour le premier acide aminé d'une protéine. Les codons **UAA, UAG et UGA** indiquent la fin de la traduction. Le codon **CUC** code pour l'acide aminé leucine.

d'un organisme.

1.2.2 Concept d'évolution

Selon la théorie du naturaliste Charles Darwin, les espèces apparaissent à la suite de séries de transformations biologiques que l'on appelle **évolution.** Au cours de l'évolution, un groupe d'individus d'une espèce peut subir des modifications qui s'accumulent pendant un nombre de générations, conduisant à la création d'une barrière reproductive au sein de la même espèce, c'est-à-dire une barrière qui empêche un groupe d'individus de se reproduire avec le reste de son espèce, et par conséquent, une nouvelle espèce se crée. Ce phénomène s'appelle la **spéciation**. Ensuite, les génomes des deux espèces soeurs évolueront indépendamment l'un de l'autre.

1.2.2.1 Homologie, paralogie et orthologie

Au cours de l'évolution, un gène peut être dupliqué (par un processus de *duplication*) ou modifié. Les gènes provenant d'un même gène ancestral sont dits **copies d'un gène**. Étant donné un ensemble de génomes, toutes les copies d'un même gène sont appelées **gènes homologues**. Ces derniers sont regroupés en une famille appelée **famille de gènes**. Au cours de l'évolution, certains de ces gènes deviennent inactifs, on les appelle **pseudogènes**. Par exemple, on recense plus de 19000 pseudogènes dans le génome humain [249]. Cette quantité est légèrement inférieure au nombre de gènes fonctionnels estimés à 22000.

Des gènes homologues de deux espèces différentes sont dits **orthologues** s'ils proviennent du même gène présent dans le dernier ancêtre commun des deux espèces. Des gènes homologues (pas nécessairement d'un même génome) sont dits **paralogues** s'ils dérivent d'une duplication.

L'évolution des espèces est la conséquence de modifications génomiques appelées **mutations**. Ces mutations affectent l'ADN à différentes échelles et sont généralement

regroupées en deux grandes classes : les mutations ponctuelles et les mutations chromosomiques [246].

1.2.2.2 Les mutations ponctuelles

Ces mutations affectent une ou quelques paires de bases (de nucléotides) adjacentes de l'ADN. On regroupe sous cette classe : l'**insertion**, la **délétion** et la **substitution**.

- L'insertion consiste à insérer une (ou quelques paires de bases) dans l'ADN.
- La délétion consiste à supprimer une (ou quelques paires de bases) de l'ADN.
- La substitution consiste à remplacer une paire de bases par une autre. Le remplacement d'une paire de bases peut entraîner un changement dans le sens du codon. Les mutations dans les codons sont généralement classifiées en **silencieuse**, **faux-sens**, **non-sens** et **neutre**.
  - La mutation silencieuse remplace un codon spécifiant un acide aminé par un codon du même acide aminé. Cette mutation n'a aucun impact sur la fonction de la protéine.
  - La mutation faux-sens remplace un codon spécifiant un acide aminé par un codon d'un acide aminé différent. La fonction initiale de la protéine résultante peut être conservée.
  - La mutation non-sens remplace un codon spécifiant un acide aminé par un codon stop (codant indiquant la fin de la traduction). La fonction initiale de la protéine résultante est généralement perdue.
  - La mutation neutre remplace un codon spécifiant un acide aminé par un codon d'un acide aminé ayant les mêmes propriétés physico-chimiques.

1.2.2.3 Les mutations chromosomiques

Ces mutations modifient un grand nombre de nucléotides, pouvant affecter donc plusieurs gènes ou le génome en entier. Ces mutations sont généralement répertoriées en deux groupes : les mutations qui changent le nombre de chromosomes dans une cellule comme par exemple la **polyploïdie** qui consiste en la duplication du génome tout entier, et les mutations chromosomiques modifiant la structure des chromosomes (voir Figure 1.4.). Ces dernières sont appelées **réarrangements** [70]. Il existe deux types de réarrangements [70] : les réarrangements déséquilibrés et les réarrangements équilibrés.

Réarrangements déséquilibrés

Les réarrangements déséquilibrés modifient la quantité de gènes (donc la quantité d'ADN) d'un segment chromosomique. On regroupe sous ce type de réarrangements la **délétion** (ou **perte**) et la **duplication**. Une délétion correspond en la suppression d'un segment de chromosome. Une duplication consiste en la duplication d'un segment de chromosome. En fonction du mécanisme biologique impliqué, que nous ne détaillons pas ici, le segment dupliqué se retrouve adjacent au segment d'origine (*i.e.* duplication en tandem), ou ailleurs dans le même chromosome, ou dans un chromosome différent.

Réarrangements équilibrés

Les réarrangements équilibrés, quant-à-eux, modifient l'ordre des gènes sur les chromosomes sans aucune perte ou duplication d'un segment d'ADN. On regroupe sous ce type de réarrangements l'**inversion**, la **transposition**, et la **translocation**. Une inversion est un réarrangement d'un segment chromosomique rompu à deux endroits puis inversé et dont les extrémités rompues ont été collées. Une transposition consiste à déplacer un segment d'un chromosome à une autre position (pas nécessairement du même chromosome). Une translocation consiste à échanger deux extrémités de deux chromosomes non homologues (*i.e.* chromosomes n'appartenant pas à la même paire).

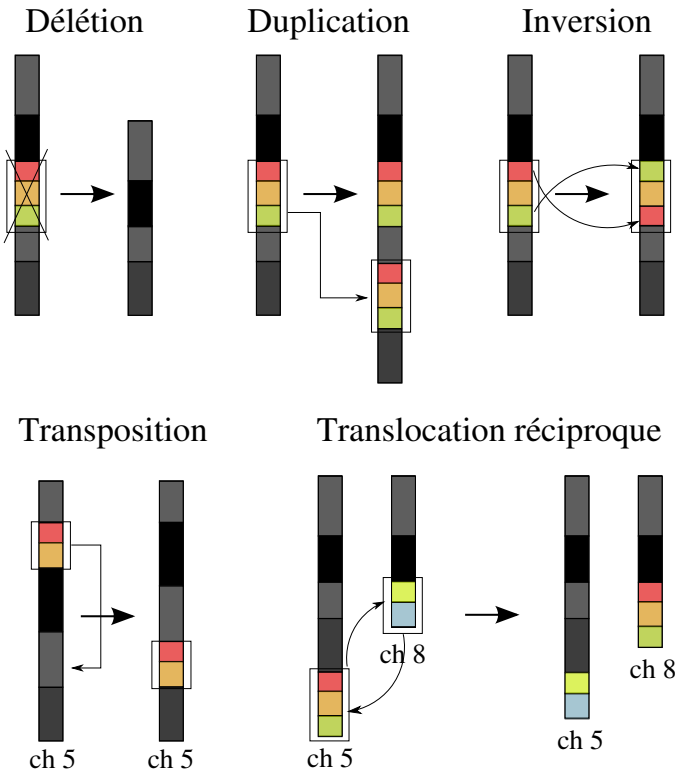


Figure 1.4 : Mutations chromosomiques modifiant la structure des chromosomes. ch8 et ch5 désignent respectivement les chromosomes 8 et 5.

Il existe en fait d'autres réarrangements impliquant des chromosomes entiers, comme la **fission** et la **fusion**. La fission consiste à couper un chromosome en deux. La fusion est l'opération inverse qui permet de réunir deux chromosomes en un seul. Il existe également chez les organismes vivants des échanges de matériel génétique entre cellules. Cela constitue le **transfert horizontal**, par opposition au transfert vertical, qui a lieu de génération en génération.

1.3 Pré-requis

1.3.1 Quelques mots sur la complexité

La théorie de la complexité est axée sur les *problèmes de décision*. Ces problèmes sont formalisés par un ensemble $\mathscr{I}$ de données en entrée (**instances**), et une question

sur l'existence d'une solution satisfaisant une certaine propriété. Il s'agit alors de déterminer, étant donnée une instance $x \in \mathscr{I}$, si la réponse est *Oui* ou *Non*. Un exemple d'un problème de décision, que l'on considère dans le chapitre 5, est le problème de *couverture minimum par sommets* (que l'on appellera CMS par souci de concision).

Exemple 1 *Étant donné un graphe $G = (V, E)$ et un entier k, le problème CMS consiste à déterminer s'il existe un sous ensemble $V' \subset V$ de taille k, appelé couverture par sommets de G, tel que chaque arête de V est incidente à au moins un sommet de V'.*

Un problème peut être résolu par un ou plusieurs algorithmes. Les algorithmes se caractérisent par une **complexité en temps** et une **complexité en espace** correspondant respectivement au temps d'exécution et à l'espace nécessaire dans le pire des cas.

Les travaux effectués dans le domaine de la théorie de la complexité ont permis d'identifier plusieurs classes de problèmes en fonction de leur complexité [203]. L'appartenance d'un problème à une certaine classe se définit par la complexité des algorithmes permettant de le résoudre. Il existe en fait un très grand nombre de classes différentes[2], mais nous nous limitons ici à présenter les deux classes suivantes : la classe **P** et la classe **NP**.

Définition 1 *Un problème est dans la classe **P** s'il peut être résolu par un algorithme déterministe en un temps polynomial par rapport à la taille de l'instance.*

La classe **P** contient l'ensemble des problèmes polynomiaux, et caractérise l'ensemble des problèmes que l'on peut résoudre facilement.

Un exemple de problème polynomial est de déterminer si un graphe est connexe ou pas. Il s'agit d'effectuer un parcours en profondeur du graphe afin de savoir si toutes les paires de sommets sont reliées par un chemin. La complexité d'un tel parcours est $O(n+m)$ (ou n est le nombre de sommets du graphe, et m est le nombre de ses arêtes). Le problème est donc dans la classe **P**.

[2]Le "zoo des complexité", consultable sur https ://complexityzoo.uwaterloo.ca/Complexity_Zoo/, recense pas moins de 498 classes de complexités différentes.

Définition 2 *Un problème est dans la classe **NP** s'il peut être résolu par un algorithme polynomial pour une machine non déterministe.*

La résolution des problèmes de **NP** peut donc nécessiter l'examen d'un grand nombre de cas (éventuellement exponentiel), et l'examen de chaque cas se fait en temps polynomial.

Il existe une relation d'inclusion entre les classes **P** et **NP** : $\mathbf{P} \subset \mathbf{NP}$. Cependant, l'une des grandes questions ouvertes est : **P = ? NP**.

Définition 3 *Un problème est **NP-difficile** s'il est au moins aussi difficile à résoudre que tous les autres problèmes de **NP**. Autrement dit, n'importe quel autre problème de **NP** peut se réduire (par transformation polynomiale) à lui. Une telle transformation est effectuée par une réduction polynomiale.*

Définition 4 *Une **réduction polynomiale** d'un problème Π à un problème Π' est un algorithme de **NP** permettant de transformer toute instance de Π en une instance de Π' telle que pour toute instance x' de Π', x' a une solution si et seulement si l'instance transformée de Π a une solution.*

Certains problèmes de **NP** sont plus difficiles à résoudre que d'autres. Les problèmes les plus difficiles de **NP** appartiennent à la classe des problèmes **NP-complets**.

Définition 5 *Un problème est **NP-complet** s'il est dans **NP** et s'il est **NP**-difficile.*

Un problème **NP-complet** célèbre est le problème CMS de couverture minimum par sommets.

1.3.1.1 Problèmes d'optimisation

Contrairement aux problèmes de décision, on parle de problèmes d'optimisation lorsqu'il y a plusieurs solutions possibles satisfaisant une certaine propriété et l'on doit trouver la solution qui optimise (maximise ou minimise) certains critères, d'où la définition suivante :

Définition 6 *Un problème d'optimisation* Π *est défini comme un quadruplet (*$\mathscr{I}$*,Sol, val, but) tel que :*

- $\mathscr{I}$ *est l'ensemble des instances de* Π.
- *pour chaque instance* $x \in \mathscr{I}$*,* $Sol(x)$ *est l'ensemble des solutions* ***réalisables*** *(ou possibles).*
- *val est une fonction associant à chaque solution possible une valeur. Pour une solution réalisable y,* $val(y)$ *est la valeur de la solution y.*
- *but* $\in \{max, min\}$*, il s'agit soit de maximiser ou soit de minimiser la fonction objective val.*

Il est possible d'associer à chaque problème d'optimisation un problème de décision dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif soit supérieure ou inférieure à une valeur donnée. Par exemple, le problème CMS défini précédemment, est associé au problème d'optimisation suivant :

Problème CMS de couverture minimum par sommets :
DONNÉES : Un graphe $G = (V, E)$.
SORTIE : Une couverture de G de cardinalité minimale.

Les problèmes d'optimisation dont la version décision est dans **NP** sont regroupés dans la classe **NPO** (NP-Optimization en anglais). La classe **NPO** est la classe des problèmes d'optimisation vérifiant les trois propriétés suivantes [19] :

1. On peut reconnaître, en temps polynomial, toutes les instances et toutes les solutions.
2. Toutes les solutions sont de taille polynomiale en la taille de l'instance correspondante. De plus, on peut reconnaître, en temps polynomial, si une solution donnée

est réalisable.

3. On peut calculer, en temps polynomial, le coût d'une solution réalisable.

1.3.2 Quelques mots sur l'approximation polynomiale

Il existe plusieurs problèmes d'optimisation où le nombre de solutions est exponentiel par rapport à la taille de l'instance. Il est donc peu probable de trouver des algorithmes efficaces pour les résoudre. Une première alternative consiste à chercher des algorithmes non polynomiaux dans le pire des cas, mais efficaces en pratique. Des méthodes ont ainsi été développées dans ce but, notamment la *programmation dynamique* et la méthode de *Séparation et Évaluation* (*Branch and Bound* en anglais).

Une seconde alternative consiste à proposer des algorithmes rapides qui, en s'autorisant une marge d'erreur dans la solution, fournissent des solutions réalisables aussi bonnes que possible. Deux types d'algorithmes ont été proposés : les algorithmes d'approximation (dont la qualité par rapport à celle du résultat optimal est bornée), et les heuristiques (dont la qualité peut être loin de la qualité du résultat optimal, mais qui tentent de fournir de bonnes solutions en pratique).

1.3.2.1 Rapport d'approximation

Il est naturel de s'intéresser, pour un problème d'optimisation donné, à la qualité des solutions fournies par les algorithmes proposés, dans le cas où l'optimalité de la solution n'est pas absolument requise. On voudrait, par exemple, avoir une garantie quant à cette qualité. L'estimation de cette dernière peut se faire par le rapport d'approximation défini ci-dessous.

Définition 7 *Soit y une solution réalisable d'une instance x d'un problème d'optimisation* Π *de* ***NPO****. On dénote par Opt(x) la* ***solution optimale*** *de l'instance x. Le rapport*

d'approximation (ou facteur d'approximation) de y sur x est défini comme suit :

$$\rho(x,y) = \frac{val(y)}{Opt(x)} \tag{1.1}$$

Ce rapport est dans $[1,\infty)$ pour un problème de minimisation, et dans $[0,1]$ pour un problème de maximisation. Il est aisé de voir que, dans les deux cas, plus ce rapport est proche de 1, meilleure est la solution.

1.3.2.2 Algorithmes d'approximation

Définition 8 *Soit* Π *un problème d'optimisation et* ρ *une fonction telle que* $\rho : \mathscr{I} \to \mathbb{R}^+$. *Étant donnée une instance* $x \in \mathscr{I}$, *un algorithme A d'approximation (ou approché) est un algorithme fournissant une solution réalisable* $A(x)$. *A est polynomial s'il fournit une solution polynomiale en la taille* $|x|$ *de l'instance.*

L'algorithme A est dit ρ*-approché si pour toute instance* $x \in \mathscr{I}$, *le rapport d'approximation de* $A(x)$ *est meilleur que* $\rho(x)$. *On dit aussi qu'un tel algorithme approche à* ρ *près le problème* Π.

La définition d'un algorithme ρ-approché nous amène à définir l'approximabilité d'un problème d'optimisation.

Définition 9 *Soit* Π *un problème d'optimisation, et* ρ *une fonction telle que* $\rho : \mathscr{I} \to \mathbb{R}^+$.

Π *est dit* ρ*-approximable s'il existe un algorithme polynomial A qui l'approche à* ρ *près (ou* ρ*-approché).*

1.3.2.3 Classe d'approximation APX

Les problèmes approximables sont généralement regroupés en classes selon leurs niveaux d'approximabilité. Nous nous limitons ici à définir la classe **APX**. Cette classe regroupe les problèmes de **NPO** pour lesquels il existe un algorithme dont le rapport d'ap-

proximation est une constante (indépendant de l'instance). Plus précisément, la classe **APX** est la classe des problèmes **NP** qui sont ρ-approximables pour un ρ constant.

Un exemple trivial d'un problème de cette classe est le problème CMS de couverture minimum par sommet, qui est 2-approximable. En effet, il existe un algorithme qui renvoie une solution au plus deux fois plus grande que la solution optimale. L'idée de base de l'algorithme consiste en le calcul d'un couplage maximal, c'est à dire un ensemble d'arêtes n'ayant pas de sommets communs, et tel que chaque arête du graphe possède au moins une extrémité commune avec une arête de cet ensemble. L'ensemble des extrémités (sommets) des arêtes de ce couplage constitue la solution de l'algorithme. Il est aisé de voir que la solution optimale a au moins une extrémité de chaque arête du couplage, et par conséquent, l'algorithme est 2-approché.

Soit Π un problème d'optimisation appartenant à la classe **APX**. Π est dit **APX-difficile** si tous les problèmes de la classe **APX** peuvent se réduire à lui sous la *réduction PTAS* (Polynomial-Time Approximation Scheme en anglais) [83] définie comme suit :

Définition 10 *Soient Π et Π' deux problèmes d'optimisation de **NPO**. On dit que Π se réduit à Π' sous la réduction PTAS s'il existe deux fonctions f et g calculables en temps polynomial telles que :*

- *si $x \in \mathscr{I}$ alors $x' = f(x) \in \mathscr{I}'$*
- *si y' est la solution de l'instance x' alors $y = g(x, y')$ est la solution de l'instance x*
- *$\forall \varepsilon > 0, \exists \gamma = \gamma(\varepsilon) > 0$, tel que si y' est la solution de l'instance x', ayant $1 + \gamma(\varepsilon)$ comme rapport d'approximation alors y est la solution de l'instance x, ayant $1 + \varepsilon$ comme rapport d'approximation.*

Toutefois, dans la pratique, une autre réduction est souvent employée afin de prouver la **APX-difficulté** d'un problème de **NPO** donné. C'est la *réduction linéaire* (ou *L-réduction* par souci de concision), introduite par Papadimitriou et al. [202] et définie ci-dessous.

Définition 11 *Soient* Π *et* Π' *deux problèmes d'optimisation de* ***NPO****. On dit que* Π *se réduit à* Π' *sous la L-réduction, s'il existe deux fonctions* f *et* g *calculables en temps polynomial, et deux constantes* α_1 *et* α_2 *telles que :*

- *si* $x \in \mathscr{I}$ *alors* $x' = f(x) \in \mathscr{I}'$
- *si* y' *est la solution de l'instance* x' *alors* $y = g(x, y')$ *est la solution de l'instance* x
- $\forall x \in \mathscr{I}$ *et* $\forall y \in Sol(f(x))$ *:*
  1. $Opt(f(x)) \leq \alpha_1 Opt(x)$
  2. $|Opt(x) - val(g(x, y'))| \leq \alpha_2 |Opt(f(x)) - val(y')|$

La L-réduction est beaucoup plus générale que la réduction PTAS dans le sens où l'existence d'une telle réduction entre deux problèmes de **NPO** implique l'existence d'une réduction PTAS entre ces deux problèmes.

Par conséquent, si l'on dispose d'une réduction de ce type entre deux problème d'optimisation Π et Π' de **NPO** et tel que Π est **APX-difficile**, on peut alors prouver la **APX-difficulté** du problème Π'.

1.4 Conclusion

Dans ce chapitre, des notions de biologie moléculaire et de complexité algorithmique, nécessaires à la compréhension des travaux réalisés dans cette thèse, ont été présentées. Nous présentons dans le chapitre suivant l'un des problèmes les plus classiques de la bioinformatique, à savoir l'alignement de séquences.

CHAPITRE 2

ÉTAT DE L'ART SUR L'ALIGNEMENT DE SÉQUENCES

2.1 Notions de base sur l'alignement de séquences

2.1.1 Introduction

L'alignement de séquences est la méthode principale utilisée en bioinformatique pour la comparaison de séquences biologiques. Cette méthode permet d'inférer les modifications impliquées dans la transformation d'une séquence en une autre. On parle généralement d'alignement *par paires* lorsqu'il s'agit de comparer deux séquences, et d'alignement *multipe* lorsqu'il s'agit d'aligner plus de deux séquences.

Un alignement de deux séquences, sous forme de chaînes de caractères (ou *résidus*), est défini comme une matrice de deux lignes dont la première (resp. la deuxième) ligne contient les caractères de la première (rep. de la deuxième) séquence dans l'ordre, augmentés d'espaces "-" appelés trous (ou "gaps"), telle qu'aucune colonne n'est formée de deux "gaps".

Les colonnes de l'alignement contenant le même caractère sur les deux lignes sont appelées des "matchs", celles qui comportent des caractères différents sont appelées des "mismatchs". Les colonnes qui comportent un espace sont appelées des "indels" ; si l'espace se trouve sur la première (resp. la seconde) ligne alors, la colonne est appelée *insertion* (resp. *délétion*).

La figure 2.1.1 montre un exemple d'alignement de deux séquences.

```
PL6_Humain    P L P S D K A P T P P G K G - A A P E S S L I T F E A A P P T L -
PL6_Mouse     P L P L E E A S T P P G K - V T V P E S S L I T L E T A - P - L L
```

Figure 2.1 : Exemple d'alignement de deux séquences.

Un des objectifs principaux d'un alignement de séquences est d'identifier des zones

ou régions *homologues* à un ensemble de séquences, *i.e.* des zones qui dérivent d'un ancêtre commun. Un alignement de séquences permet également de reconnaître une signature commune à un ensemble de séquences, d'étudier un ensemble de protéines apparentées afin d'extraire des motifs d'une importance structurelle ou fonctionnelle d'une même famille. Enfin, l'alignement de séquences est à la base de la reconstruction d'histoires évolutives entre les séquences.

L'alignement de séquences peut être *global*, c'est-à-dire qu'il couvre la totalité des séquences alignées, ou *local*. Dans ce cas, l'alignement identifie des régions ou sous-séquences similaires sans se soucier de la longueur des séquences ou de l'organisation des régions.

La suite de cette partie est organisée comme suit : dans la section 2.1.2, nous présentons les éléments nécessaires à la comparaison des différents alignements. Ensuite, dans la section 2.1.3, nous citons brièvement les différents algorithmes d'alignement de séquences.

2.1.2 Comparaison d'alignements de séquences

Afin de pouvoir déterminer si un alignement est meilleur qu'un autre, un *score* est attribué à chaque alignement. Un tel score est choisi de telle sorte à refléter la qualité de l'alignement. Le *problème d'alignement de séquences* consiste alors à déterminer l'alignement de meilleur score.

Généralement, les modèles de score utilisés sont basés sur le principe de *score additif*, dans lequel le score de l'alignement est égal à la somme des scores des paires de caractères des séquences alignées. Ceci revient à supposer que les mutations à chaque site de l'alignement se produisent de façon indépendante. Dans ces modèles, on associe un score à chaque type de colonne, et l'objectif est de maximiser (ou de minimiser) le score de l'alignement. Le meilleur alignement (*i.e.* alignement optimal) est donc celui dont le score est le plus élevé (ou le plus bas).

L'utilisation d'un modèle de score nécessite la définition d'une *matrice de substitution* donnant les différents scores de substitution d'un caractère par un autre dans l'alignement. En complément de cette matrice, un modèle de pénalité des "gaps" est nécessaire pour pouvoir associer des scores aux "indels". Plusieurs matrices de substitution existent selon les séquences étudiées. Dans le cas de séquences protéiques (*i.e.* séquences d'acides aminés), les matrices les plus utilisées sont les matrices **PAM** [88] et **BLOSUM** [143]. Dans le cas de séquences d'ADN, les matrices de substitution sont en général des modèles markoviens [114].

2.1.3 Algorithmes d'alignement de séquences

Plusieurs méthodes algorithmiques exactes existantes permettent d'aligner deux séquences selon des modèles évolutifs différents. La plupart d'entre elles se basent sur la programmation dynamique.

Contrairement à l'alignement par paires, la plupart des méthodes dédiées à l'alignement multiple sont des heuristiques, étant donné que les méthodes exactes deviennent rapidement inutilisables pour un nombre suffisamment grand de séquences.

2.1.3.1 Alignement par paires

Les algorithmes d'alignement par paires les plus connus sont l'algorithme de Needleman et Wunsch [191] et l'algorithme de Smith et Waterman [232].

L'algorithme de Needleman et Wunsch est considéré comme la référence pour l'alignement global de deux séquences. Pour deux séquences S_1 et S_2 de longueurs respectivement n et m, l'algorithme construit une matrice A de dimension nm telle que la valeur de $A(i, j)$ (pour $1 \leq i \leq n$, $1 \leq j \leq m$) correspond au score optimal de l'alignement des préfixes $S_1[1..i]$ et $S_2[1..j]$.

Le calcul du score repose sur l'utilisation d'un modèle de pénalité des "gaps" et d'une matrice de substitution. Dans le cas d'un modèle de pénalité à fonction linéaire, la

matrice se remplit de la façon suivante :

$$A(i,j) = max \begin{cases} A(i-1,j-1) + W(S_1[i], S_2[j]) & (1) \\ A(i-1,j) + \lambda & (2) \\ A(i,j-1) + \lambda & (3) \end{cases}$$

où $W(S_1[i], S_2[j])$ est le coût d'une substitution de $S_1[i]$ par $S_2[j]$, et λ est le coût d'une "indel".

Les initialisations pour cet algorithme sont données par :

$$A(0,j) = \lambda xj$$
$$A(0,j) = \lambda xi$$

Afin de pouvoir construire le ou les alignements optimaux, l'algorithme mémorise, pour chaque valeur $A(i,j)$ calculée, le chemin emprunté pour son calcul. La complexité en espace et en temps de cet algorithme est $O(nm)$.

L'algorithme de Smith et Waterman, conçu pour l'alignement local de séquences, est basé sur l'algorithme de Needleman et Wunsch avec une seule différence essentielle. N'importe quelle case de la matrice peut être considérée comme point de départ pour le calcul du score. Pour identifier le plus long segment homologue entre les deux séquences alignées, il faut parcourir toute la matrice en sens inverse et trouver la case du plus grand score.

Recherche dans des bases de données

L'alignement de deux séquences permet également de déterminer la similarité entre une séquence appelée *séquence requête* et des séquences répertoriées dans une base de données. Bien que les algorithmes précédents peuvent être utilisés dans ce cadre, leur complexité quadratique les rendent inutilisables sur des bases de données volumineuses. Plusieurs heuristiques rapides ont été alors proposées, parmi lesquelles nous citons Fasta [205], Blast [8], Gapped-Blast [9] (mise à jour de Blast), PatternHunter [185], BlastZ [223] et YASS [194].

Ces heuristiques sont basées sur des méthodes de filtrages. D'abord, la requête est divisée en sous-séquences (segments, mots), définissant les points d'ancrage de la recherche. Ces points d'ancrage sont recherchés dans les séquences par une méthode exacte, et seront ensuite étendus afin de déterminer des régions potentiellement intéressantes (régions conservées). Cela permet de filtrer la base de données pour ne retenir que les séquences contenant ces régions. Enfin, les régions potentiellement intéressantes sont réunies par un algorithme d'alignement.

2.1.3.2 Alignement multiple

Les algorithmes d'alignement multiple sont très nombreux. On peut envisager la généralisation des méthodes d'alignement de deux séquences. Cependant, ces méthodes deviennent lentes quand le nombre de séquences augmente. De plus, le problème de trouver le meilleur alignement multiple d'un ensemble de séquences est **NP-Complet**.

Certains algorithmes se basent sur l'algorithme de Needleman et Wunsch comme par exemple l'algorithme MSA (*Multiple Sequence Alignment* en anglais) [69], qui est basé sur une restriction de la zone de calcul autour de la diagonale de la matrice. Cet algorithme est à la base d'un autre algorithme appelé DCA (*Divide and Conquer Alignment* en anglais) [100].

La classe des méthodes la plus connue pour l'alignement multiple est celle des *algorithmes progressifs*. Ces algorithmes consistent à aligner progressivement les séquences en suivant un ordre bien défini. La différence entre ces algorithmes repose sur l'ordre choisi pour aligner les séquences. Bien que l'alignement obtenu n'est pas garanti d'être optimal, ces algorithmes ont l'avantage d'être rapides et donnent en général des résultats raisonnables.

Parmi les algorithmes progressifs les plus utilisés, nous citons ClustalW [241], T-Coffee [196] et Muscle [105]. L'algorithme ClustalW est l'un des algorithmes d'alignement multiple les plus rapides.

D'autres algorithmes peuvent être considérés, notamment l'algorithme Dialign [189], Multiz [41], 3D-Coffee [198] pour l'alignement de séquences protéiques, et l'algorithme MAFFT [159]. Dans ce dernier, les acides aminés sont décrits par leurs polarités et leurs volumes (propriétés physico-chimiques) et les séquences sont transformées en séquences de valeurs numériques. L'algorithme MAFFT fait appel à un algorithme appelé *transformée de Fourier rapide* afin de repérer les segments homologues entre les séquences alignées et d'accélérer le temps de calcul.

Enfin, nous citons l'algorithme ProbCons [98] qui est basé sur un *modèle de Markov caché*. L'algorithme ProbCons est l'un des meilleurs algorithmes d'alignement multiple en raison de la qualité de ses résultats. Toutefois, l'algorithme n'est pas conçu pour l'alignement d'un très grand nombre de séquences, en raison de sa complexité élevée en temps et en espace.

Étant donné le nombre important de méthodes et d'algorithmes d'alignement développés, une question importante est de savoir quel algorithme et quels critères devraient être utilisés pour une famille de séquences donnée. Selon les recommandations de Risler et Tagu [212], pour l'alignement de 2 à 100 séquences protéiques de longueur maximum de 10000 résidus, il est préférable d'utiliser l'algorithme ProbCons, T-Coffee ou Muscle. Pour l'alignement de 100 à 500 séquences ou plus, il est préférable d'utiliser l'algorithme Muscle ou MAFFT. Enfin, pour un petit nombre de séquences de longueur supérieure à 20000 résidus, il est préférable d'utiliser l'algorithme ClustalW .

Pour plus de détails sur les nouveaux progrès concernant les algorithmes d'alignement multiple, le lecteur peut consulter [195].

2.1.4 Conclusion

Cette première partie de ce chapitre constitue une brève introduction aux algorithmes classiques d'alignement de séquences. L'une des limites de ces algorithmes est qu'ils deviennent rapidement inutilisable pour l'alignement de très longues séquences telles

que les séquences de génomes complets, et par conséquent, le recours à de nouvelles méthodes d'alignement est nécessaire. La deuxième partie de ce chapitre est consacrée à ce sujet.

2.2 Alignement de génomes complets

2.2.1 Introduction

Il existe actuellement plus de 10000 génomes complètement séquencés[1]. Ce nombre important nécessite de développer des méthodes efficaces de génomique comparative. Plusieurs études biologiques sont dédiées, par exemple, à la comparaison de structures et de fonctions de protéines ou à l'annotation fonctionnelle des génomes. Ces recherches reposent sur des approches bioinformatiques et plus particulièrement sur la comparaison de génomes.

La comparaison de génomes complets peut se faire sur la base : 1) de la séquence complète de nucléotides ; 2) de l'ordre des gènes ou d'autres blocs constitutifs du génome ; 3) du protéome.

Dans cette partie, nous considérons uniquement la comparaison des génomes représentés par leurs séquences complètes de nucléotides. Dans ce cas, l'alignement de séquences est à la base de cette comparaison.

L'alignement de génomes complets pose certains problèmes qui empêchent l'utilisation des algorithmes usuels (comme l'algorithme de Needleman et Wunsch, et l'algorithme de Smith et Waterman). En effet, la taille des séquences considérées rend impossible l'utilisation d'algorithmes exacts. À titre d'exemple, le génome humain compte presque 3 milliards de nucléotides. Un algorithme en temps quadratique est clairement inutilisable sur de telles séquences, d'où la nécessité de recourir à des heuristiques.

Plusieurs algorithmes d'alignement de séquences ne considèrent que les séquences

[1] Avril 2014, http ://www.the-scientist.com/ ?articles.view/articleNo/39742/title/Sequencing-the -Tree-of-Life/

codantes [175]. Cependant, ces dernières ne représentent qu'une petite partie du génome. À titre d'exemple, le génome humain compte seulement 5% de régions codantes [248]. L'alignement de séquences intergéniques est plus difficile car les séquences sont moins conservées, elles peuvent donc contenir de long "indels" et des régions qui ne présentent aucune similarité.

Enfin, si les algorithmes classiques fonctionnent bien sur des génomes dont l'architecture (ou l'organisation) n'a pas été modifiée, ils ne sont pas adaptés aux génomes réarrangés ou ayant subi des duplications. Bien qu'il existe des algorithmes d'alignement qui prennent en compte les réarrangements et/ou les duplications [173, 245], ils ne sont pas utilisables sur de très longues séquences ou des génomes complets en raison de leur importante complexité de calcul. De plus, ils permettent seulement de détecter de petits réarrangements locaux et des duplications en tandem.

Nous décrivons, dans la section 2.2.2, la principale stratégie partagée par de nombreux algorithmes d'alignement de génomes complets (comme MUMmer [89], MGA [147], AVID [56], LAGAN [57], Multi-LAGAN [57], Shuffle-LAGAN [59], MAVID [56], Mauve [85]). Nous présentons ensuite, dans la section 2.2.3, les algorithmes d'alignement par paires et multiple les plus utilisés, qui tiennent compte des réarrangements et dont la plupart se basent sur cette stratégie.

2.2.2 Stratégie générale d'alignement de génomes complets

Dans cette section, nous présentons la stratégie générale partagée par plusieurs algorithmes d'alignement de génomes complets. Cette stratégie porte le nom de *stratégie basée sur les ancres*, et est considérée comme la référence pour l'alignement de génomes complets en raison de sa complexité sous-quadratique [1].

La stratégie basée sur les ancres est divisée en 5 étapes : (1) pré-traitement ; (2) sélection de graines ; (3) chaînage ; (4) récursivité ; (5) alignement final. Cette stratégie est illustrée sur la figure 2.2.

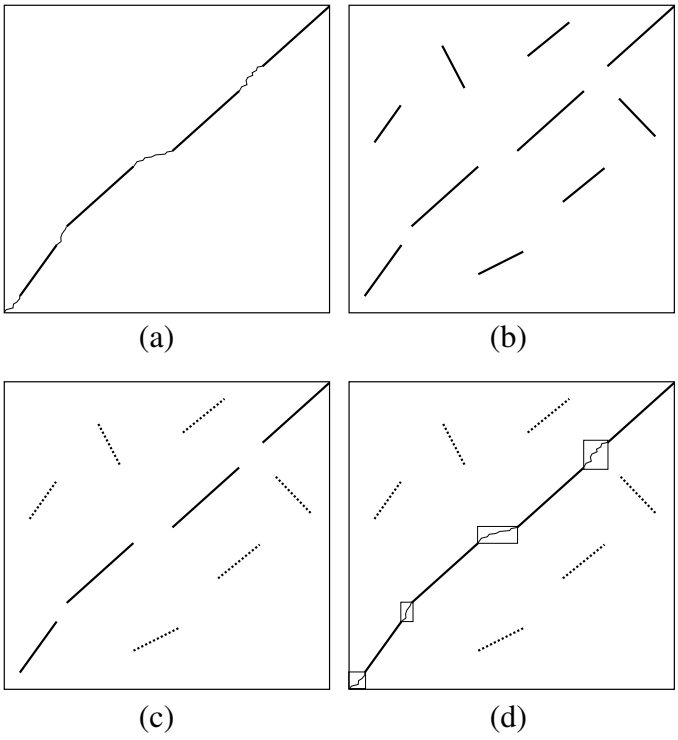


Figure 2.2 : Illustration de la stratégie basée sur les ancres. (a) L'alignement final de deux génomes, obtenu par la stratégie basée sur les ancres. (b) Étape de sélection de graines. Les segments du graphique correspondent aux régions similaires des deux génomes alignés. Les segments en pointillés correspondent aux graines éliminées lors de cette étape. (c) Étape de chaînage. Dans cet exemple, la chaîne d'acres obtenue est formée uniquement de fragments colinéaires. (d) Étape de récursivité dans laquelle sont alignées les régions entre les ancres.

2.2.2.1 Pré-traitement

Le pré-traitement permet de filtrer les séquences pour éliminer les segments ayant peu d'intérêt comme les régions d'ADN de faible complexité, *i.e.* les séquences répétées appelées *micro-satellites*, ou les *répétitions dispersées* qui sont susceptibles de perturber les autres étapes.

2.2.2.2 Sélection de graines

Cette étape est commune à tous les algorithmes d'alignement de génomes complets. Elle consiste à détecter des segments conservés entre les génomes en effectuant un alignement local. Les segments similaires peuvent être des mots exacts en commun, presque exacts, ou des alignements locaux avec très peu d'"indels". Ils sont de tailles variables et doivent parfois respecter certaines conditions de non-chevauchement. Ces segments sont appelés *fragments* et sont considérés comme des *graines* car certains d'entre eux seront à la base de la construction de l'alignement de génomes.

Cette étape est la plus importante car les choix effectués pour créer les alignements influencent directement le résultat final.

2.2.2.3 Chaînage

L'objectif de cette étape est de filtrer les graines déterminées à l'étape précédente afin de garder seulement celles qui serviront à la construction de l'alignement des génomes. Les graines éliminées peuvent êtres dues à des duplications (appelées *faux positifs*), et les graines sélectionnées sont appelées *ancres*.

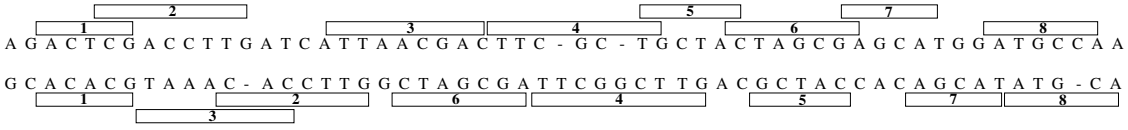
La sélection des ancres dépend des méthodes et des critères que ces méthodes souhaitent satisfaire. L'un des critères est de considérer des fragments (ou ancres) colinéaires ou non-colinéaires. Deux fragments sont *colinéaires* si leurs segments sont dans le même ordre sur les génomes. Par exemple, à la figure 2.3(a), les fragments 1 et 3 sont colinéaires, tandis que les fragments 1 et 2 sont colinéaires et se chevauchent.

Deux approches sont utilisées à cette étape [244]. L'approche de *chaînage colinéaire* qui ne considère pas les réarrangements, et l'approche de *chaînage avec réarrangements*. Notez que, dans le cadre de cette thèse, ce sont les alignements avec réarrangements qui nous intéressent tout particulièrement. Cependant, avant de détailler les méthodes en présence de réarrangements, il est nécessaire d'introduire les concepts sans réarrangements. De plus, par souci de simplicité, nous considérons uniquement le cas de comparaison de

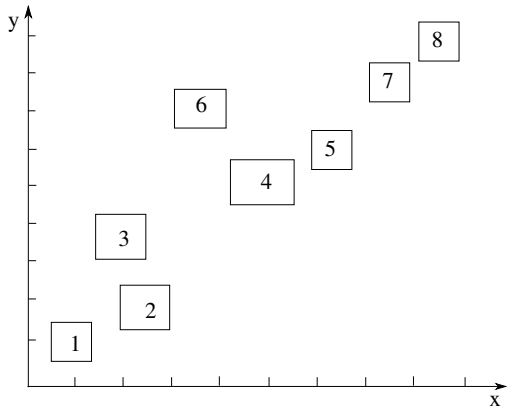
deux génomes.

Ces deux approches consistent à former une chaîne d'ancres, et se basent sur la représentation des fragments déterminés à l'étape précédente sous forme de rectangles. Un rectangle B d'un fragment f, qui se retrouve dans les deux génomes, se caractérise par deux points $beg(B)$ et $end(B)$ représentant respectivement son coin inférieur gauche et son coin supérieur droit. Les coordonnées ($beg(B).x$, $beg(B).y$) du point $beg(B)$ correspondent aux deux extrémités droites de f, et les coordonnées ($end(B).x$, $end(B).y$) du point $end(B)$ correspondent aux deux extrémités gauches de f.

Un exemple d'une collection de fragments et sa représentation sous forme de rectangles est montré à la figure 2.3.



(a)



(b)

Figure 2.3 : Représentation d'une collection de fragments sous forme de rectangles. Chaque axe représente l'un des deux génomes comparés. (a) Les 8 fragments des deux génomes comparés. (b) Les 8 rectangles correspondant aux 8 fragments montrés en (a).

Dans cette représentation, les notions de colinéarité et de chevauchement sont définies comme suit :

Définition 12 *Soient deux fragments f_1, f_2, et les deux rectangles B_1, B_2 qui leurs correspondent.*

Soit la relation de ***dominance*** $\ll$ *définie entre les deux rectangles B_1 et B_2 comme suit : $B_1 \ll B_2$ (i.e. B_1 est dominé par B_2) si et seulement si $end(B_1).x \leq beg(B_2).x$ et $end(B_1).y \leq beg(B_2).y$. Si $B_1 \ll B_2$ alors les deux fragments f_1 et f_2 sont colinéaires et ne se chevauchent pas. On dit alors que B_1 est dominé par B_2. Si B_1 n'est pas dominé par B_2 et B_2 n'est pas dominé par B_1 alors B_1 et B_2 sont* ***incomparable****.*

À titre d'exemple, dans la figure 2.3, les fragments 1 et 3 sont colinéaires car les deux rectangles qui leurs correspondent vérifient $1 \ll 3$. Il en est de même pour les fragments 2 et 4.

Cette représentation permet de définir une chaîne de fragments colinéaires non-chevauchants (ou une chaîne tout court), comme suit :

Définition 13 *Une chaîne C est une séquence de rectangles $B_1, B_2, \ldots, B_k$ reliés par la relation de dominance $\ll$ (i.e. $B_i \ll B_{i+1}$, $\forall\ 1 \leq i < k$).*

Chaque rectangle est associé à un poids qui reflète le degré de similarité des segments du fragment qui lui correspond. Si l'on note le poids d'un rectangle B par $w(B)$ alors, le poids de la chaîne C sera $W(C) = \sum_{i=1}^{k} w(B_i)$.

Ainsi, le problème que l'approche de chaînage colinéaire vise à résoudre s'appelle *problème de chaînage colinéaire*, et est défini comme suit :

Définition 14 *Soient $F = \{f_2, f_3, \ldots, f_{n-1}\}$ un ensemble de fragments et $\beta = B_2, B_3, \ldots, B_{n-1}$ l'ensemble des rectangles correspondant. Soient B_1 et B_n deux rectangles tels que $B_1 \ll B_i \ll B_n$ $\forall i$, $1 < i < n$, et $w(B_1) = w(B_n) = 0$.*

Le problème de chaînage colinéaire consiste à trouver, dans β, une chaîne C de poids maximum (i.e. $W(C)$ est maximale) qui commence par B_1 et se termine par B_n.

Plusieurs algorithmes ont été proposés afin de résoudre ce problème. Certains d'entre eux se limitent à la comparaison de deux génomes [152, 155, 244], alors que d'autres considèrent plusieurs génomes [1, 115, 190].

Afin de pouvoir détecter les réarrangements dans l'alignement de deux génomes, on assouplit la contrainte de colinéarité des fragments. Autrement dit, l'ordre des fragments sera imposé uniquement sur l'un des deux génomes. De plus, les segments constituant un fragment peuvent avoir des orientations opposées, c'est-à-dire que l'un des deux segments est le complément de l'inverse de l'autre segment). Avant de donner une formulation du problème que l'approche de chaînage avec réarrangements vise à résoudre, on introduit la notion de *conflit* entre rectangles.

Définition 15 *Deux rectangles B_i, B_j sont en **conflit** si leurs projections se chevauchent sur au moins l'un des deux axes.*

Dans la figure 2.4, les rectangles 1 et 3 sont en conflit car leurs projections sur l'axe des x se chevauchent. Les rectangles en conflit correspondent aux fragments dont les segments se chevauchent sur l'un ou sur les deux génomes alignés. Les rectangles 2 et 3, quant à eux, ne sont ni en conflit, ni munis de la relation de dominance. Cette configuration correspond à un événement de transposition. D'autres événements peuvent être détectés tels que l'inversion. Ceci nécessite d'introduire la notion suivante :

Définition 16 *On appelle **fragment inverse**, correspondant à un événement d'inversion, un fragment dans lequel les deux segments ont des orientations opposées. Dans le cas contraire, le fragment est dit **direct**.*

La représentation des fragments sous forme de rectangles telle qu'elle a été définie précédemment ne permet pas de différencier entre un fragment inverse et un fragment direct (*i.e.* observer un événement d'inversion). On ajoute pour cela des diagonales à l'intérieur des rectangles : une *diagonale principale* pour un rectangle correspondant à

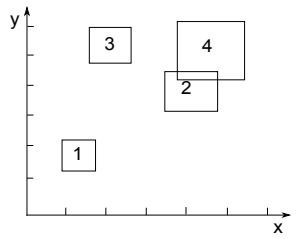


Figure 2.4 : Plusieurs configurations possibles des rectangles dans le plan. Les rectangles 1 et 3 sont en conflit car leurs projections sur l'axe des x se chevauchent. Il en est de même pour les rectangles 2 et 4, 3 et 4. Les rectangles 1 et 2, 1 et 4 sont munis de la relation de dominance. Les rectangles 2 et 3 ne sont ni en conflit ni munis de la relation de dominance.

un fragment direct, et une *anti-diagonale* pour un rectangle correspondant à un fragment inverse. Un exemple est donné à la figure 2.5.

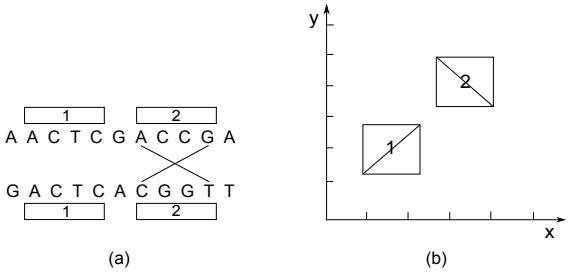


Figure 2.5 : Représentation des deux types de fragments sous forme de rectangles. (a) Deux fragments de deux génomes. Le fragment 1 est un fragment direct, tandis que le fragment 2 est un fragment inverse. (b) Les deux rectangles correspondant aux deux fragments des deux génomes. Le rectangle avec une diagonale principale correspond à un fragment direct, et le rectangle avec une anti-diagonale correspond à un fragment inverse.

Dans cette nouvelle représentation, une chaîne de fragments avec réarrangements est définie comme suit :

Définition 17 *Un ensemble de rectangles $B = \{B_1, B_2, \ldots B_k\}$ forme une chaîne avec réarrangements si les rectangles de B sont indépendants. Autrement dit, s'il n'existe pas une paire de rectangles de B en conflit.*

Une chaîne de fragments avec réarrangements a également un poids qui est la somme des poids de ses rectangles. Ainsi, le problème que l'approche de chaînage avec réarrangement vise à résoudre s'appelle *problème de chaînage avec réarrangements*, et est défini comme suit :

Définition 18 *Le problème de chaînage avec réarrangements consiste à trouver une chaîne de rectangles avec réarrangements, de poids maximum.*

Un exemple d'une chaîne avec réarrangements, de poids maximum est montré à la figure 2.6.

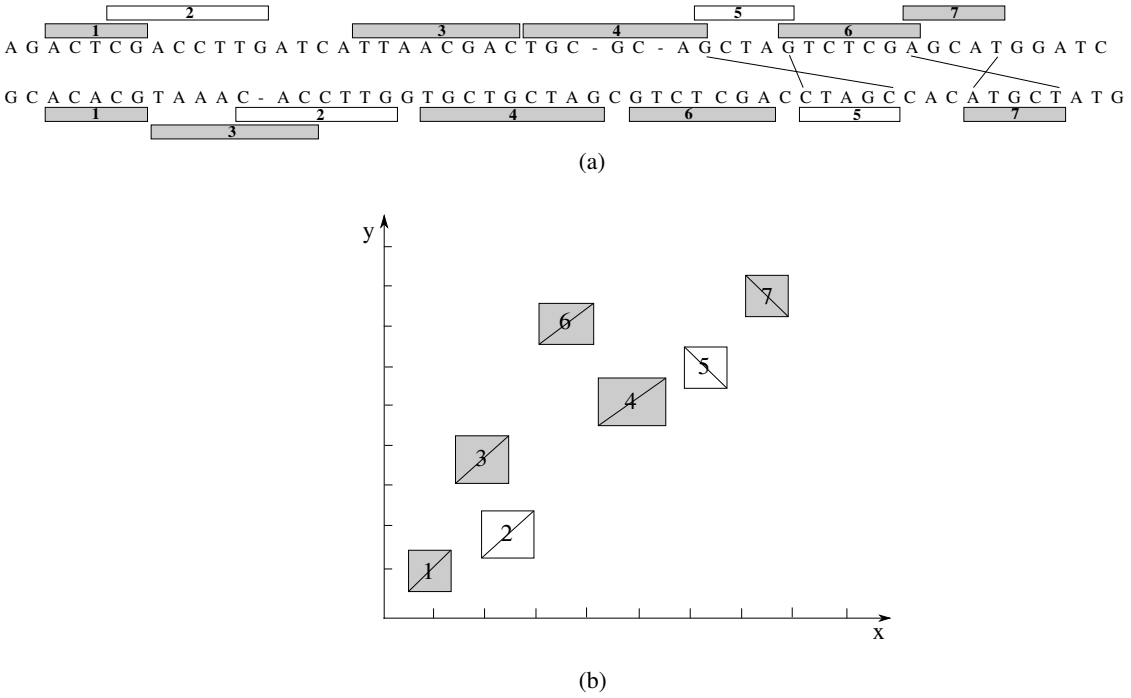


Figure 2.6 : Une chaîne avec réarrangements, de poids maximum. (a) Les 8 fragments de deux génomes comparés. (b) La chaîne avec réarrangements de poids maximum (rectangles colorés en gris) correspondant aux fragments des deux génomes.

En fait, ce problème peut être reformulé en problème de stable[2] de poids maximum

[2]Un stable (aussi appelé ensemble indépendant) dans un graphe est un ensemble de sommets deux à deux non adjacents

dans un graphe d'intersection[3]. Ce dernier a été prouvé **NP-Complet** [25].

Une heuristique, nommée *chaînage glocal* (*glocal chaining* en anglais), a été proposée pour ce problème et utilisée dans l'algorithme Shuffle-LAGAN [59] que l'on présentera dans la section 2.2.3.1. Cette heuristique est conçue pour la comparaison de deux génomes seulement.

2.2.2.4 Récursivité

Cette étape consiste à ré-exécuter un certain nombre de fois les deux étapes précédentes sur les régions des génomes qui ne sont pas considérées dans des ancres. Afin de pouvoir relier les extrémités de chaque ancre, on assouplit les critères de qualité des alignements locaux (*i.e.* les filtres utilisés à la première étape).

2.2.2.5 Alignement final

L'objectif de cette étape est de créer un alignement final des deux génomes comparés en reliant les ancres, en utilisant si nécessaire de grands "indels".

2.2.3 Algorithmes d'alignement de génomes complets

Plusieurs algorithmes ont été proposés dans le cadre de l'alignement de génomes complets. Nous proposons dans le tableau 2.I une liste (qui n'est pas exhaustive) des algorithmes d'alignement de génomes complets existants. Ces algorithmes sont ordonnés par année de publication.

La figure 2.7 montre une classification de presque tous les algorithmes d'alignement cités dans le tableau 2.I. Cette classification a été faite par Treangen et al. [242]. Les algorithmes sont regroupés principalement selon leurs types (paires ou multiple), et selon s'ils considèrent les réarrangements ou pas.

[3]Un graphe d'intersection est un graphe représentant les intersections d'une famille d'ensembles finie donnée

Algorithme	Année de publication
DBA [153]	1999
MUMmer [89]	1999
GLASS [28]	2000
PipMaker [224]	2000
WABA [164]	2000
OWEN [213]	2002
MGA [147]	2002
MUMmer v3.0 [90]	2002
AVID [56]	2003
LAGAN [57]	2003
Multi-LAGAN [57]	2003
Shuffle-LAGAN [59]	2003
CHAINNET [163]	2003
MALGEN [117]	2003
MAVID [56]	2003
POA [130]	2004
ABA [211]	2004
MULAN [199]	2004
TBA [41]	2004
EMAGEN [92]	2004
Mauve [85]	2004
Chainer [2]	2004
GATA [193]	2005
ACANA [150]	2005
M-GCAT [242]	2006
Magic [235]	2006
M-PECAN [204]	2008
CoCoNUT [3]	2008
GR-Aligner [79]	2009
ProgressiveMauve [87]	2010
Mugsy [16]	2011
LS-BSR [215]	2014

Tableau 2.I : Liste des algorithmes d'alignement de génomes complets, ordonnés par année de publication.

Dans la suite, nous explicitons quelques algorithmes considérant les réarrangements.

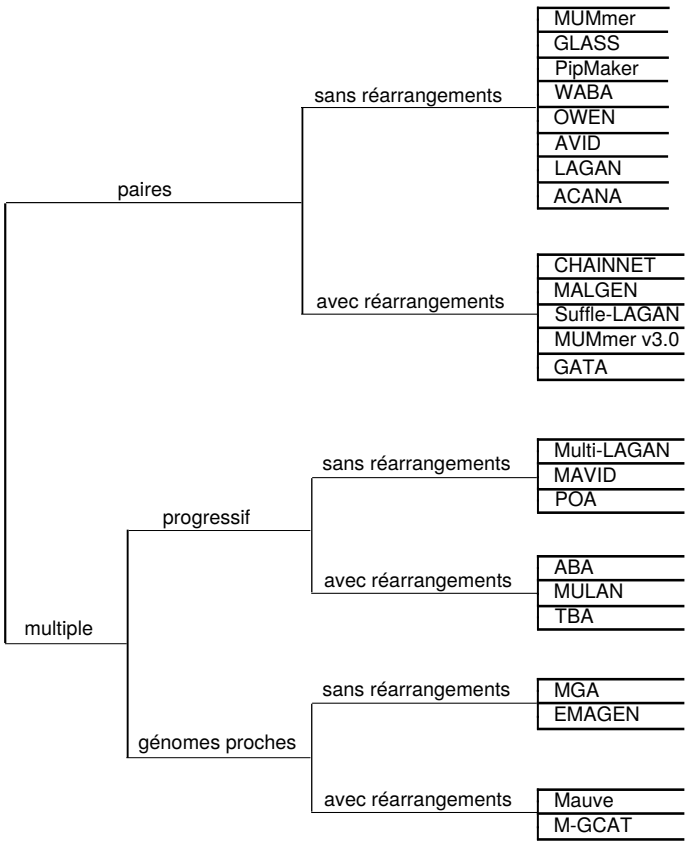


Figure 2.7 : Classification des algorithmes d'alignement de génomes complets. Les algorithmes sont regroupés principalement selon leurs types (paires ou multiple), et selon s'ils considèrent les réarrangements ou pas.

2.2.3.1 Algorithmes d'alignement par paires

LAGAN et Shuffle-LAGAN

LAGAN

LAGAN utilise l'algorithme d'alignement local CHAOS pour l'identification de fragments appelés *(k,c)-graine*. Une (k,c)-graine est un segment de taille k, commun aux deux génomes comparés avec au plus c différences. Dans certains ouvrages [2, 244], une (k,c)-graine est appelée *inexact k-mers*.

Une fois ces fragments identifiés, ils sont ensuite chaînés afin de créer des aligne-

ments locaux. Deux fragments sont étendus si la distance qui les sépare dans les deux génomes est inférieure à un certain seuil. Pour deux (k,c)-graines données, la distance qui les sépare dépend de la distance qui sépare leurs segments dans la première et dans la deuxième séquence. Puis, LAGAN utilise l'algorithme de programmation dynamique éparse décrit dans [111], pour trouver la chaîne des fragments colinéaires non-chevauchants (ancres), de poids maximum.

Enfin, les régions entre les ancres seront alignées par l'algorithme de Needleman et Wunsch. L'algorithme LAGAN a été utilisé pour la comparaison de 12 espèces de Vertébrés.

Shuffle-LAGAN

Shuffle-LAGAN est une extension de LAGAN qui considère les réarrangements comme l'inversion, la translocation et la translocation inversée.

La première étape de Shuffle-LAGAN consiste en l'utilisation de l'algorithme CHAOS pour l'identification des (k,c)-graines qui seront étendues afin de créer des alignements locaux (*i.e.* les fragments).

Ensuite, l'algorithme emploie l'heuristique *chaînage glocal* qui consiste à créer, à partir des fragments identifiés à l'étape précédente, une **chaîne monotone**, *i.e.* une chaîne dans laquelle l'ordre des fragments est non décroissant sur la première séquence sans aucune restriction sur la deuxième séquence. Cependant, les fragments ne doivent pas se chevaucher sur la deuxième séquence.

Lors du chaînage d'un fragment f_i avec le fragment f_j qui le précède dans la chaîne, on attribue une pénalité qui dépend de si les fragments f_i et f_j sont de même type (*i.e.* les deux sont directs ou inverses) ou pas, et si f_i se situe après (*i.e.* dans le même ordre sur les deux séquences) ou avant f_j sur la deuxième séquence. En résumé, ces 4 configurations définissent 4 types de pénalités : "gap" régulier (fragments de même type, dans le même ordre), inversion (fragments de types différents, dans le même ordre), transloca-

tion (fragments de même type, d'ordre inverse sur la deuxième séquence), translocation inversée (fragments de types différents, d'ordre inverse sur la deuxième séquence).

La somme de ces pénalités constitue la distance d'édition entre les deux génomes comparés. La détermination de la chaîne monotone peut se faire en temps $O(n \log n)$ par l'emploi de l'algorithme de programmation dynamique éparse [111].

Une fois la chaîne monotone déterminée, elle est ensuite divisée en plusieurs sous-chaînes ou *blocs localement colinéaires LCB* (*Locally collinear blocks* en anglais). Les fragments de chaque bloc ont la particularité d'être strictement croissants ou strictement décroissants sur la deuxième séquence. De plus, chaque bloc est maximal, c'est à dire que l'on ne peut pas l'étendre à gauche ou à droite.

La figure 2.8 montre un exemple d'une chaîne monotone optimale déterminée par l'algorithme Shuffle-LAGAN, ainsi que ses différents blocs localement colinéaires.

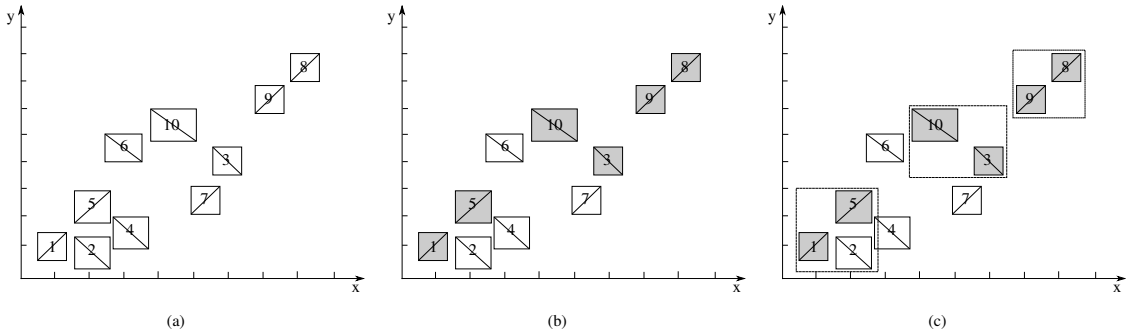


Figure 2.8 : Les différentes étapes de détermination des LCB de la chaîne monotone des différents fragments de deux génomes alignés. (a) L'ensemble des rectangles correspondant aux fragments des deux génomes. (b) La chaîne monotone optimale (rectangles en gris) des fragments des deux génomes. (c) Les différents LCB (rectangles à l'intérieur d'une boite) de la chaîne monotone déterminée en (b). Les rectangles à l'intérieur d'un bloc ne sont pas en conflit, tandis que les rectangles de différents blocs peuvent se chevaucher sur la deuxième séquence, comme les rectangles 2 et 7.

Enfin, la dernière étape de l'algorithme consiste à aligner les différents blocs localement colinéaires par l'algorithme LAGAN.

CHAINNET

CHAINNET est un algorithme d'alignement de génomes complets qui considère également les réarrangements comme l'inversion, la translocation et les duplications.

Les fragments considérés par l'algorithme sont les alignements locaux sans "indels" produits par BLASTZ.

L'algorithme utilise, ensuite, la structure des *arbres k-d*, qui consiste en la partition des données dans un espace à k dimensions, afin de pouvoir trouver les fragments colinéaires qui peuvent être chaînés (*i.e.* les ancres). L'algorithme tient compte de l'orientation des fragments.

Enfin, CHAINNET produit plusieurs chaînes et attribue un score à chacune d'elles. Ces chaînes sont placées par ordre décroissant de leurs scores puis traitées les unes après les autres. À chaque itération, l'algorithme marque les positions du génome qui sont couvertes par les ancres d'une chaîne. Une position donnée ne peut être couverte que par une ancre d'une seule chaîne. Si la chaîne considérée couvre des positions déjà marquées alors l'algorithme garde uniquement les parties de cette chaîne qui couvrent des positions qui ne sont pas encore marquées. L'algorithme utilise la structure des *arbres rouge et noir* pour pouvoir garder trace des positions du génomes déjà marquées.

De plus, si une chaîne couvre des positions d'un gap d'une chaîne déjà traitée, celle-ci est alors considérée comme enfant d'une telle chaîne. Ainsi, CHAINNET forme une hiérarchie de chaînes. Cependant, le détails sur la détermination de l'ensemble des chaînes n'a pas été montré dans l'article [163]. L'algorithme a été utilisé pour la comparaison des génomes de l'homme et la souris.

2.2.3.2 Algorithmes d'alignement multiple

Dans cette partie, nous présentons certains algorithmes d'alignement multiple de génomes complets les plus connus, qui considèrent les réarrangements. Certains d'entre

eux sont des extensions des algorithmes par paires de génomes comme Multi-LAGAN. Les deux algorithmes Multi-LAGAN et Mauve sont des algorithmes progressifs adaptés à la comparaisons de séquences évolutivement éloignées.

Multi-LAGAN

Multi-LAGAN est une généralisation de LAGAN à l'alignement de plusieurs séquences. L'algorithme se résume en trois étapes principales.

La première étape consiste à calculer tous les alignements de toutes les paires de séquences par l'algorithme LAGAN.

Dans la deuxième étape, l'algorithme utilise l'arbre phylogénétique (ou arbre guide) des séquences afin de pouvoir décider de la séquence à incorporer dans l'alignement multiple. Chaque alignement de deux ou de plusieurs séquences est considéré à nouveau comme une nouvelle "séquence-multiple" (ou consensus) S_m. L'algorithme détermine, ensuite, toutes les chaînes colinéaires de poids maximum entre la nouvelle séquence S_m et toutes les autres "séquences-multiples", par l'emploie de l'algorithme LIS (*Longest Increasing subsequence* en anglais) de recherche de la plus longue sous-suite strictement croissante [132]. Les ancres considérées dans le calcul d'une chaîne de deux séquences-multiples S_i et S_j sont l'union des ancres déterminées lors de l'alignement des paires des séquences formant les deux "séquences-multiples" S_i et S_j.

La dernière étape de l'algorithme est une étape de raffinement pendant laquelle l'algorithme retire successivement une séquence S de l'alignement multiple, puis la réaligner avec l'alignement restant par l'algorithme LAGAN jusqu'à ce que le score de l'alignement converge. Certaines régions de la séquence S qui améliorent le score de l'alignement multiple seront considérées comme nouvelles ancres. L'algorithme Multi-LAGAN a été utilisé pour la comparaison de 12 espèces de vertébrés.

Mauve

La première étape de Mauve consiste à déterminer des segments conservés appelés *multi-MUMs* (*multiple Maximal Unique Matches* en anglais). Les *multi-MUMs* sont des segments communs à tous les génomes comparés. Ils ont la particularité d'être uniques, c'est à dire qu'il existe une seule occurrence de chacun d'eux dans le génome. Ceci permet de réduire la sensibilité de l'algorithme dans les régions conservées répétées. Chaque multi-MUMs est maximal, c'est à dire qu'on ne peux pas l'étendre à gauche ou à droite. Afin de pouvoir déterminer les *multi-MUMs*, l'algorithme emploie une méthode de hachage similaire à celle utilisée dans l'algorithme GRIL d'identification de régions colinéaires [86]. Ces multi-MUMs serviront également à la création d'une matrice de distances utilisée pour la construction d'un arbre phylogénétique (ou arbre-guide) par la méthode de Neighbour-joining. Cet arbre est utilisé dans la dernière étape de l'algorithme.

L'algorithme divise, ensuite, l'ensemble des fragments déterminés à l'étape précédente en un nombre minimum de blocs localement colinéaires ou LCB (voir section 2.2.3.1), par l'emploie de l'algorithme de Sankoff [217] d'analyse de points de cassure.

Mauve associe un poids à chaque LCB, qui dépend de la longueur des multi-MUMs, et ne garde que les LCB de poids supérieur à un certain seuil. Ainsi, l'algorithme fusionne les LCB adjacents.

La figure 2.9 montre un exemple d'identification des LCB des différents multi-MUMs de trois génomes, ainsi que les LCB retenus par l'algorithme.

Afin de pouvoir déterminer les régions homologues entre les LCB et celles à l'intérieur de chaque LCB, l'algorithme ré-exécute les étapes précédentes plusieurs fois, sur les régions qui n'ont pas été considérées à la première étape, jusqu'à ce qu'aucune ancre (ou multi-MUMs) ne puisse plus être détectée ou jusqu'à ce que la longueur des régions soit inférieure à un certain seuil. La longueur des ancres dépend des longueurs des différentes régions considérées. Ainsi, les régions entre les différents LCB seront concaténées

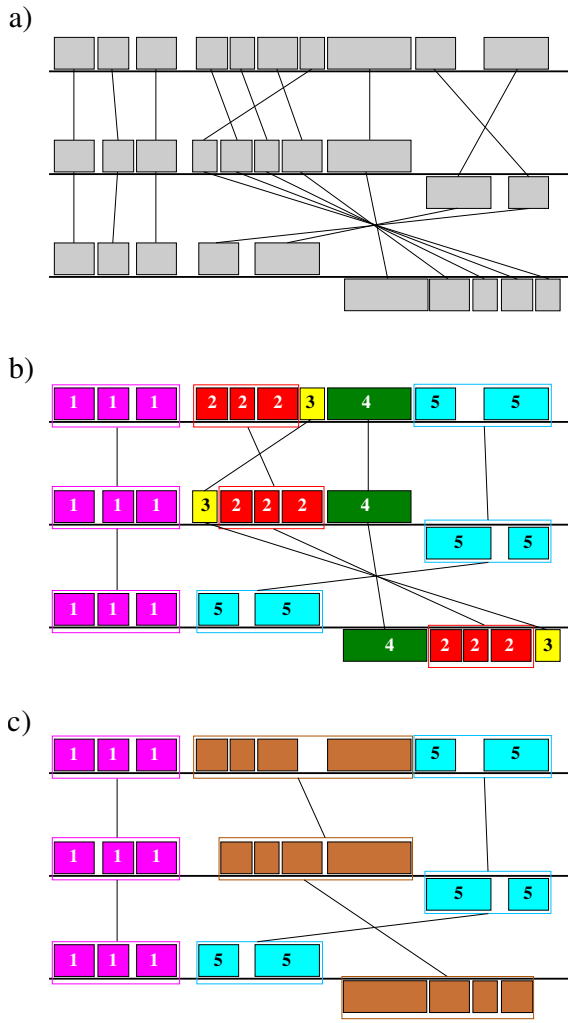


Figure 2.9 : Les différentes étapes d'identification des LCB de trois génomes alignés. (a) Identification des multi-MUMs. Chaque multi-MUM est composé de 3 blocs connectés. (b) Partionnement des multi-MUMs en un nombre minimum de LCB. Chaque LCB est représenté par un numéro unique. (c) Élimination des LCB de poids inférieur à un certain seuil. Il s'agit du LCB 3. Les deux LCB adjacents 2 et 4 peuvent donc être fusionnés en un seul LCB.

afin de pouvoir détecter de nouveaux multi-MUMs.

Enfin, les régions restantes à l'intérieur des LCB et de longueur inférieure à 10kb sont alignées par l'algorithme progressif ClustalW, en se basant sur l'arbre guide déterminé à la première étape. L'algorithme a été utilisé pour la comparaison de 9 génomes de

bactéries de la famille des *Entérobactéries*.

L'une des limites de l'algorithme Mauve est qu'il ne permet pas d'identifier des régions conservées par un sous-ensemble des génomes comparés. Par la suite, une extension de l'algorithme nommée ProgressiveMauve [87] a été proposée, capable de détecter les régions conservées uniquement par un sous-ensemble des génomes alignés. Cette nouvelle version permet également d'aligner un ensemble plus large de génomes que la version précédente, et est plus performante en terme d'exactitude. Cependant, ProgressiveMauve est plus lent que Mauve et consomme plus de mémoire.

M-GCAT

M-GCAT est à la fois un algorithme et outil de visualisation conçu pour la comparaison de plusieurs génomes de grandes tailles et évolutivement proches. L'algorithme permet de détecter les réarrangements comme l'inversion et la transposition.

M-GCAT détermine les multi-MUMs sur les génomes de façon récursive. Tout d'abord, l'algorithme recherche un premier ensemble des multi-MUMs de longueur supérieure à un certain seuil puis, la recherche est étendue récursivement sur les régions entre les ancres en considérant à chaque fois des valeurs décroissantes de la taille des régions et du seuil des longueurs des multi-MUMs autorisées.

Ensuite, une étape de filtrage est utilisée pour éliminer certains multi-MUMs considérés comme des "faux matchs". Seulement les multi-MUMs de longueur supérieure à un certain seuil seront retenus.

Enfin, l'algorithme regroupe les ancres qui se trouvent à une certaine distance maximale (en terme de nucléotides) en blocs colinéaires similaires aux LCB utilisés dans l'algorithme MAUVE.

Afin de pouvoir évaluer les performances de l'algorithme en terme de temps d'exécution, M-GCAT et Mauve ont été comparés sur 8 ensembles de génomes d'entérobactéries (famille de bactéries). Les résultats révèlent que M-GCAT est plus rapide que

Mauve mais consomme plus de mémoire [242].

2.2.4 Comparaison d'algorithmes d'alignement de génomes complets

Les différents algorithmes d'alignement (par paires ou multiple) de génomes complets existants produisent souvent des alignements différents même lorsqu'ils sont appliqués sur les mêmes séquences génomiques [93], conduisant les utilisateurs et les chercheurs à se demander quel alignement est considéré "correct" d'un point de vue biologique. Le développement de méthodes d'évaluation des performance des algorithmes d'alignement de génomes complets et de la qualité des alignements produits par ces algorithmes est donc nécessaire [39].

Une étude comparative a été réalisée par Chen et Tompa [72], dans laquelle les algorithmes TBA, MAVID, Multi-LAGAN et Pecan ont été comparés sur un jeu de données de 28 génomes de vertébrés. Les résultats de la comparaison montrent que Pecan donne les meilleurs résultats en terme qualité de l'alignement. Cependant, les auteurs concluent que la construction d'un alignement multiple de génomes reste une tâche difficile pour les régions non-codantes et pour les espèces éloignées. De plus, le développement de méthodes de comparaison de plusieurs algorithmes sur des jeux de données de grandes tailles est nécessaire.

2.2.5 Conclusion

Dans cette partie, nous avons présenté les algorithmes d'alignement de séquences de génomes complets, les plus connus et dont la majorité partage une même stratégie basée sur les ancres. Nous avons, plus particulièrement, décrit les algorithmes qui permettent de considérer certains réarrangements comme l'inversion et la translocation. Les algorithmes présentés ont été classifiés en plusieurs catégories : alignement par paires ou multiple, et selon s'ils ont été conçus pour l'alignement de séquences évolutivement éloignées (algorithmes progressifs) ou évolutivement proches. Toutefois, tous ces algo-

rithmes demeurent des heuristiques non bornées, sans aucune garantie théorique sur le résultat produit, dont l'incertitude est amplifiée par la taille des séquences de génomes manipulées.

Dans le prochain chapitre, nous nous intéressons à la comparaison de génomes, non pas au niveau de la séquence de nucléotides, mais à un niveau plus élevé considérant une représentation plus compacte des génomes, en blocs structuraux, dont les plus importants sont les gènes.

CHAPITRE 3

APERÇU DES MÉTHODES DE RÉARRANGEMENTS ET DE RECONSTRUCTION D'ORDRES DE GÈNES

3.1 Introduction

Dans le chapitre précédent, la comparaison de génomes était basée sur l'alignement de leurs séquences nucléotidiques. Dans ce chapitre, nous comparons les génomes à un niveau plus élevé : nous observons l'arrangement des séquences identifiées comme homologues entre les génomes. Ces séquences sont désignées par des marqueurs qui sont généralement les gènes, et les génomes sont représentés sous forme de suites de gènes.

Les premiers travaux algorithmiques sur la comparaison de génomes, en se basant sur l'ordre des blocs structurants (ci-après on parlera de gènes), se sont limités à comparer des génomes contenant exactement les mêmes gènes sans aucun gène dupliqué (*i.e.* un seul exemplaire ou copie de gène par famille) [26, 139, 160, 162, 221, 253]. Les génomes étaient donc modélisés par des permutations. Ce modèle a permis le calcul de plusieurs mesures de distances, à savoir la distance de points de cassure, les distances de réarrangements (comme les inversions, les translocations, les transpositions, les échanges de blocs et DCJ), et également les intervalles communs et les intervalles conservés. Dans le cas de génomes possédant des gènes dupliqués, un prétraitement doit, tout d'abord, être effectué afin de ne garder qu'un seul exemplaire de chaque gène.

Dans le cas des distances de réarrangements, on considère une ou plusieurs opérations et on cherche à trouver un *scénario*, c'est à dire une séquence de telles opérations, permettant de transformer un génome en un autre. Généralement, on se base sur un critère de parcimonie, c'est à dire que l'on cherche la plus courte séquence d'opérations (scénario optimal permettant de transformer une permutation en une autre). Afin de se rapprocher des réalités biologiques, des méthodes ont été proposées en associant un coût

(ou pondération) par rapport aux événements considérés [23, 24, 42, 46, 234]. En effet, certains événements sont beaucoup plus fréquents que d'autres au sein des espèces étudiées. De plus, certains événements sont très fréquents chez certaines espèces et rares chez d'autres [120, 174, 243].

Le scénario obtenu fournit des informations pertinentes sur l'évolution des espèces des génomes comparés depuis leur ancêtre commun. Ainsi, à partir d'un ensemble de génomes, il est possible de reconstruire un arbre phylogénétique, des ordres ancestraux et des scénarios expliquant les liens entre les génomes. Ce problème s'appelle le *problème de la grande phylogénie* (*Large Phylogeny Problem* en anglais).

Dans ce chapitre, nous étudions un cas particulier de ce problème, connu sous le nom du *problème de la petite phylogénie* (*Small Phylogeny Problem* en anglais). Étant donné un arbre phylogénétique (*i.e.* arbre[1] expliquant l'histoire évolutive des espèces, dont la topologie est connue à l'avance) dont les génomes sont représentés aux feuilles, ce problème consiste à déterminer les ordres de gènes des génomes ancestraux identifiés par les noeuds internes de l'arbre, de telle sorte à minimiser le poids total de l'arbre, *i.e.* la somme des scores de toutes les branches.

Le problème de la petite phylogénie a été prouvé **NP-difficile** pour la plupart des distances de réarrangements [168]. Des heuristiques ont été alors proposées dans le but de résoudre ce problème de manière rapide et efficace.

Ce chapitre est organisé comme suite. Dans la section 3.2, nous donnons une représentation formelle des génomes. Cette représentation sera également considérée dans les prochains chapitres. Puis, nous présentons, dans la section 3.3, les principales distances entre deux génomes comme la distance de points de cassure et les différentes distances de réarrangements (translocation, inversion, transposition, échange de blocs et DCJ). Pour chaque distance, nous discutons brièvement de sa complexité de calcul, les algorithmes proposés et les différentes variantes, dans le cas de génomes ayant le même

[1] Dans le cadre de cette thèse, nous considérons uniquement des arbres binaires

ou différents contenus en gènes, avec ou sans gènes dupliqués.

Ensuite, dans la section 3.4, nous présentons certains modèles spécifiques à la comparaison de génomes possédant des gènes dupliqués.

Enfin, nous présentons, dans la section 3.5, les algorithmes de reconstruction d'ordres ancestraux dans un arbre phylogénétique donné. Dans un premier temps, nous présentons le problème de la médiane dont la solution est d'une importante clé pour l'inférence d'ordres ancestraux. Ensuite, nous présentons les deux principales méthodes de reconstruction d'ordres ancestraux dans un arbre phylogénétique. La première est la méthode de *steinérization* qui se base essentiellement sur le problème de la médiane. La seconde est la méthode de *recherche locale*, considérée comme généralisation de la méthode de *steinérization*. Ces deux méthodes ont l'avantage de pouvoir intégrer n'importe quel algorithme de reconstruction d'ordres ancestraux. Finalement, nous citons brièvement les algorithmes partageant ces deux méthodes.

3.2 Représentation d'un génome

Soit un alphabet Σ dont chaque caractère représente une famille de gènes. Nous représenterons un génome par un ensemble de séquences (l'ensemble des chromosomes) constituées de caractères de Σ. Chaque caractère (ou gène) peut apparaître 0, 1 ou plusieurs fois dans un génome. Un caractère qui existe en un seul exemplaire dans un génome est qualifié de "**singleton**". De plus, étant donné qu'un gène est transcrit à partir d'un brin particulier de l'ADN, chaque gène porte un signe ($+$ ou $-$; on omet généralement le signe $+$) en fonction de son orientation de transcription. Le nombre de gènes du génome constitue sa **taille**.

Soit par exemple $\Sigma = \{a, b, c\}$, la séquence $X = ((a_1\ c_1\ a_2), (b_1\ c_2))$ représente un génome multichromosomique (possédant deux chromosomes) de taille 5, constitué de deux copies du gène a, deux copies du gène c et une copie du gène b. Le gène b est un "singleton".

Soit $X = x_1 x_2 ..... x_n$ une séquence (ou chromosome) de n gènes. On appelle séquence **inverse** de X la séquence $-X = -x_n ... - x_2 - x_1$.

Dans le cas d'un chromosome **circulaire** $X = x_1 x_2 ..... x_n$, le gène x_1 est considéré suivre le gène x_n. Un chromosome qui n'est pas circulaire est **linéaire**.

3.3 Distances entre deux génomes

3.3.1 Distance de points de cassure

Soit deux génomes X et Y, que l'on veut comparer. Un point de cassure (*breakpoint* en anglais) désigne deux gènes qui sont consécutifs (ou adjacents) dans X mais pas dans Y. Dans le cas de gènes signés, on considère, en plus, un point de cassure lorsque deux gènes consécutifs dans les deux génomes n'ont pas leurs signes conservés (par exemple, ab dans X et $a - b$ dans Y). On considère également l'adjacence entre le premier gène et le dernier gène dans le cas de génomes circulaires.

La distance de points de cassure entre deux génomes X et Y est le nombre de points de cassure dans X relativement à Y, ou de façon équivalente dans Y relativement à X. Le calcul de cette distance peut se faire de façon triviale en temps linéaire. Cette distance a été introduite dans l'article de Kececioglu et Sankoff [162] comme une approximation de la distance d'inversion (voir section 3.3.3). Elle est considérée comme une mesure de dissimilarité qui compte les différences entre deux génomes.

La figure 3.1 donne une illustration de cette distance dans le cas de gènes signés.

La section suivante est consacrée à la présentation des différentes distances de ré-

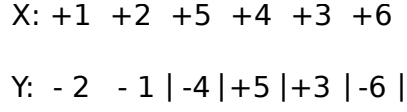


Figure 3.1 : Exemple de points de cassure entre deux génomes X et Y. Chaque point de cassure est représenté par une barre verticale. Il y a 5 points de cassure entre X et Y.

arrangements. Avant cela, il est nécessaire d'introduire le *graphe de points de cassure*, utilisé pour le calcul de certaines distances.

3.3.2 Graphe de points de cassure

Le graphe de points de cassure est un graphe représentant les adjacences entre les gènes des deux génomes comparés. Dans ce graphe, un gène "a" dans l'orientation positive (resp. dans l'orientation inverse) est représenté par deux sommets a_t et a_h (resp. a_h et a_t) dans cet ordre, où a_t et a_h sont appelées respectivement la queue (*tail* en anglais) et la tête (*head* en anglais) du gène "a", et représentent les extrémités de ce dernier. Ainsi, une adjacence signifie que deux extrémités de deux gènes sont consécutives. Une extrémité d'un gène qui n'est pas adjacente à une extrémité d'un autre gène est appelée *télomère*.

Pour représenter les adjacences entre les gènes des deux génomes, on utilise deux types d'arêtes (généralement, des arêtes de deux couleurs différentes) : des arêtes de couleur noire pour représenter les adjacences dans un génome, et des arêtes de couleur grise pour représenter les adjacences dans l'autre génome.

Dans le cas de génomes linéaires multichromosomiques, il faut tenir compte des adjacences aux télomères. Par exemple, le génome multichromosomique linéaire $X = ((2\ 3), (1\ -5\ -4))$ compte 7 adjacences dont les trois adjacences $2_h\ 3_t$, $1_h\ 5_h$, $5_t\ 4_h$ et les quatre adjacences aux télomères 2_t, 3_h, 1_t et 4_t. Pour simplifier, on peut ne pas représenter ces adjacences dans le graphe. Dans ce cas, un sommet se trouvant à l'extrémité d'un chromosome dans les deux génomes n'est incident à aucune arête, un sommet se trouvant à l'extrémité d'un chromosome dans un génome sans l'autre est incident à une arête, et les autres sommets sont chacun incident à deux arêtes.

La figure 3.2 présente le graphe de points de cassure pour les deux génomes linéaires multichromosomiques $X = ((1\ 2), (3\ 4\ 5))$ et $Y = ((1\ -2), (3\ -4\ 5))$.

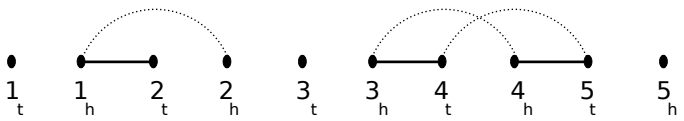


Figure 3.2 : Graphe de points de cassure pour les deux génomes linéaires multichromosomiques $X = ((1\ 2), (3\ 4\ 5))$ et $Y = ((1\ -2), (3\ -4\ 5))$

3.3.3 Distances de réarrangements

Dans cette section, on considère une distance D entre deux génomes comme le nombre minimum d'opérations de D permettant de transformer un génome en un autre. On parle généralement du problème de tri par D, dont l'objectif est de trouver la suite minimale d'opérations de D permettant de transformer un génome en un autre. De plus, les gènes des deux génomes à comparer sont généralement numérotés de telle sorte que l'un des deux soit réduit à la permutation identité Id.

Distance de translocation et distance d'inversion

Le problème de tri par translocations a été introduit par Kececioglu et Ravi [161] dans le cas de gènes signés, où un algorithme 2-approché a été proposé. Par la suite, des algorithmes polynomiaux ont été proposés [35, 136, 201]. En parallèle à cela, un algorithme en temps linéaire pour le calcul de la distance de translocation entre deux génomes a été proposé [179]. Cependant, dans le cas de gènes non-signés, le calcul de cette distance a été prouvé **NP-difficile** [262].

Notez qu'il est possible de modéliser les translocations par des inversions si l'on concatène les chromosomes des deux génomes comparés, en les représentant ainsi par deux permutations (voir Figure 3.3 pour un exemple).

La distance d'inversion a été introduite par Sankoff [216] dans le cas de gènes signés. Le premier algorithme polynomial pour le problème de tri par inversions a été présenté par Hannennalli et Pevzner [139]. L'algorithme proposé est de complexité $O(n^4)$, et

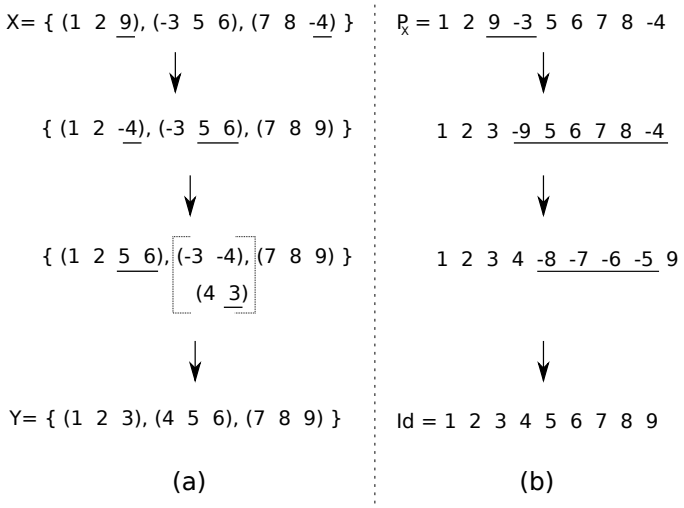


Figure 3.3 : Exemple de modélisation de translocations par des inversions. Une barre horizontale indique soit les deux extrémités (des deux chromosomes) à changer, dans le cas d'une translocation, ou soit les gènes concernés par une inversion. (a) Séquence de translocations entre deux génomes multichromosomiques X et Y. Par définition, les deux chromosomes $(-3\,-4)$ et $(4\ 3)$ sont considérés identiques. (b) Une séquence d'inversions entre les deux permutations P_X et Id correspondant respectivement aux deux génomes X et Y.

utilise une modélisation des génomes sous forme de graphe de points de cassure. En se basant sur ce graphe, les auteurs ont prouvé que la distance d'inversion entre deux permutations P et Id peut se calculer par la formule suivante :

$$D(P,I_d) = n + 1 - c(P,I_d) + h(P,I_d) + f(P,I_d)$$

où n est le nombre de gènes, $c(P,I_d)$ est le nombre de cycles du graphe de points de cassure, $h(P,I_d)$ et $f(P,I_d)$ sont deux fonctions représentant des composantes plus complexes du graphe. Ces composantes ne seront pas détaillées ici car elles sont rarement observées sur des données réelles. Pour une description détaillée, nous référons le lecteur à l'article [139]

Plusieurs améliorations de l'algorithme de Hannennalli et Pevzner ont été ensuite publiées. Kaplan et al. [156] ont proposé un algorithme plus rapide en temps quadratique

puis, Bader et al. [20] ont proposé une amélioration en temps linéaire pour le calcul de la distance d'inversion seulement (sans donner le scénario).

Ensuite, Ozery-Flato et Shamir [200] ont proposé des corrections et améliorations pour l'algorithme de Hannenalli et Pevzner [139]. Les auteurs se sont aussi posés la question de l'existence d'un algorithme sous-quadratique pour le calcul de la distance d'inversion. En se basant sur les travaux de Kaplan et Verbin [157], Tannier et al. [238, 239] répondent à la question en proposant un algorithme en temps $O(n^{3/2}\sqrt{\log n})$. Par la suite, Swenson et al. [233] ont amélioré cette complexité en proposant un algorithme en temps $O(n \log n)$.

Récemment, le tri par de petites inversions (de taille au plus 2 ou 3) a été considéré [125]. Galvao et al. [126] ont proposé un algorithme polynomial dans le cas des inversions de taille au plus 2, et un algorithme 5-approché dans le cas des inversions de taille au plus 3.

Dans le cas de gènes non-signés, le calcul de la distance d'inversion entre deux génomes a été prouvé **NP-difficile** [66]. Le meilleur algorithme d'approximation développé est de facteur 1.375 [38].

D'autres études ont été menées sur des variantes du problème, comme le tri par inversions des préfixes (le bloc à inverser est un préfixe de la séquence), connu sous le nom de *retournement de crêpes* (*Pancake Flipping Problem* en anglais), introduit en 1975 par Dweighter [102] puis, considéré à plusieurs reprises [96, 121, 128, 145, 208]. Ce problème a été récemment prouvé **NP-difficile** par Bulteau et al. [63].

D'autres travaux ont porté sur le tri par inversions dans des génomes possédant des gènes dupliqués [77, 165, 166, 209], dans des génomes ayant différents contenus en gènes (avec ou sans gènes dupliqués) en considérant des opérations d'insertion et de délétion [107, 186], sur le tri par inversions pondérées [17, 18, 24, 29, 234] et sur le tri par inversions des préfixes et suffixes [181, 182, 214]. Très récemment, une autre variante du tri par inversions a été considerée par Lintzmayer et al. [183]. Ces derniers

ont montré que le tri par inversions pondérées des préfixes seulement ou des préfixes et suffixes est $log^2 n$-approximable, dans le cas de gènes signés ou non-signés.

Distance de transposition et distance d'échange de blocs

Le problème de tri par transpositions a été introduit par Bafna et Pevzner [26] (voir la figure 3.4(a) pour un exemple d'une transposition). La complexité du problème a été prouvée **NP-difficile** par Bulteau et al. [64]. Parallèlement à la distance de transposition, une version plus générale, connue sous le nom d'échange de blocs (*blocks interchange* en anglais), dans laquelle les blocs à échanger ne sont pas nécessairement consécutifs (voir figure 3.4(b) pour un exemple), a été introduite par Christie [76], où un algorithme en temps quadratique a été proposé. Par la suite, des algorithmes améliorant la complexité de ce dernier ont été proposés [151, 180].

Certains auteurs se sont intéressés aux développement d'heuristiques, d'algorithmes d'approximation et de bornes pour le problème de tri par transpositions [27, 78, 94, 95, 112, 116, 123, 131, 133, 140, 170]. Le meilleur algorithme d'approximation développé est de facteur 1.375 [110].

Des variantes du problème ont également été considérées comme le tri par transpositions des préfixes introduit par Dias et Meidanis [97], où un algorithme 2-approché a été proposé, puis étudié par d'autres [74, 96, 171, 230]. Le tri par transpositions et le tri par transpositions des préfixes dans des génomes possédant des gènes dupliqués ont également été considérés [77, 82, 209, 229]. Chitturi et al. [74] ont étudié le problème de tri par transpositions de séquences binaires. Les auteurs ont montré que ce problème est **NP-complet**, et ont également proposé une borne supérieure ($n - \log_{\frac{9}{2}} n$) pour la distance de transposition des préfixes entre deux permutations.

D'autres études ont porté sur le tri par transpositions pondérées dans des génomes possédant des gènes dupliqués [10, 11, 46, 141] et récemment, sur le tri par transpositions des préfixes et suffixes [181, 182, 214].

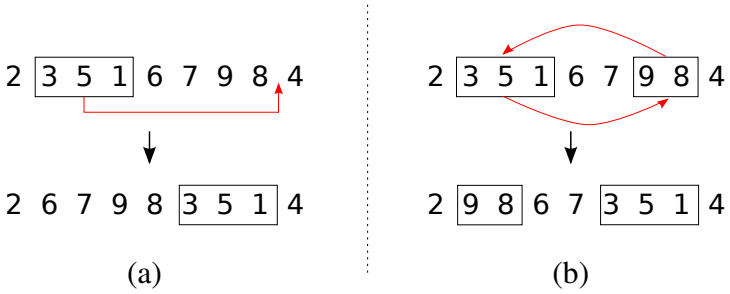


Figure 3.4 : Exemple de transposition et d'échange de blocs. (a) Transposition déplaçant le bloc (351) à la position indiquée par la flèche rouge. Cette opération consiste à échanger deux blocs consécutifs. Dans cet exemple, ce sont les blocs (351) et (6798). (b) Échange des deux blocs (351) et (98).

Très récemment, Galvao et al. [126] ont considéré le tri par de petites inversions et transpositions. La taille L_I des inversions considérées est au plus 2 ou 3, tandis que pour les transpositions, la somme L_T des tailles des deux blocs consécutifs à échanger est au plus 2 ou 3. Les auteurs ont proposé un algorithme polynomial dans le cas où la valeur de L_I et L_T est au plus 2, et un algorithme 3-approché dans le cas où la valeur de L_I et L_T est au plus 3. Les problèmes de tri par transpositions pondérées des préfixes seulement, et des préfixes et suffixes ont été étudiés par Lintzmayer et al. [183]. Ces derniers ont montré que ces deux variantes sont $log^2 n$-approximables.

3.3.4 Distance DCJ

L'opération (ou le modèle) DCJ (*Double Cut and Join*) est une opération de réarrangement plus générale qui permet de modéliser les translocations, les inversions, les transpositions et les échanges de blocs. Aussi appelée *2-break*, cette opération consiste à faire deux coupures dans un génome (ou un chromosome) puis recoller les bouts générés de n'importe quelle manière.

Un événement d'inversion nécessite une seule opération DCJ, tandis qu'un événement de transposition ou d'échange de blocs nécessite deux opérations DCJ et le passage par un chromosome circulaire intermédiaire (ou temporaire).

La figure 3.5 montre un exemple de modélisation d'un événement d'inversion et d'un événement de transposition par des opérations DCJ.

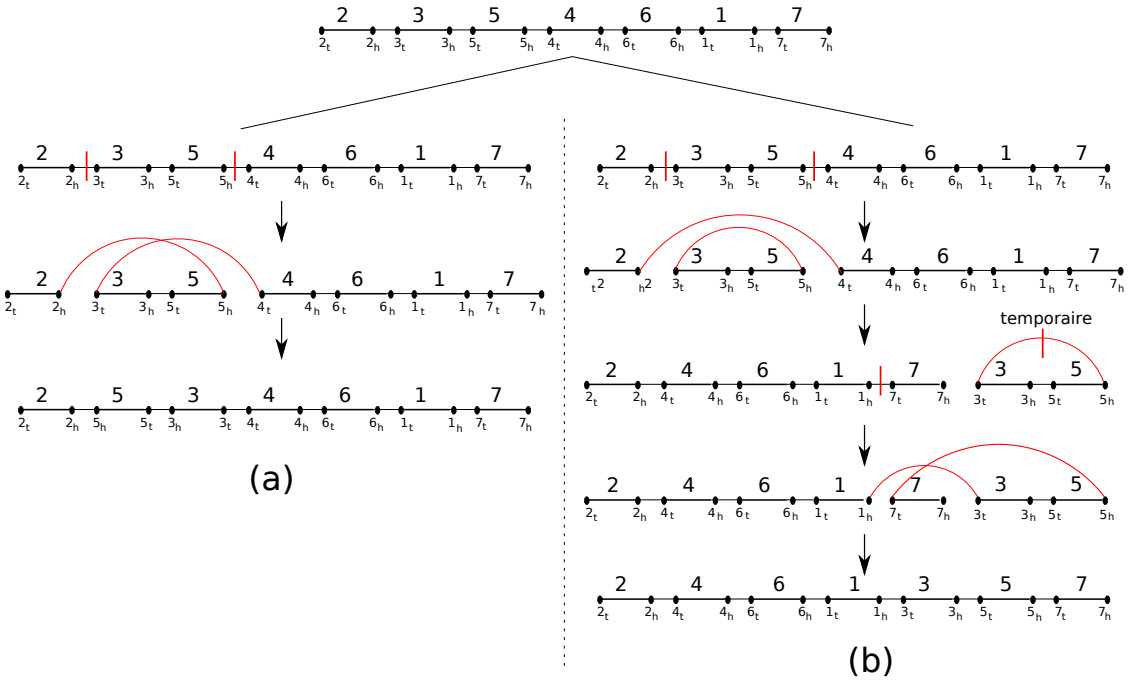


Figure 3.5 : Exemple de réarrangements modélisés par des opérations DCJ. Les coupures et les liens entre les extrémités des chromosomes sont représentés respectivement par des barres verticales rouges et des barres courbes rouges. (a) Modélisation d'une inversion du segment (35) par une opération DCJ. (b) Modélisation d'une transposition du segment (35) par deux opérations consécutives DCJ en passant par un chromosome temporaire circulaire.

La distance DCJ a été introduite par Yancopoulos et al. [253] puis, simplifiée par Bergeron et al. [36], où un algorithme en temps linéaire a été proposé pour le calcul de cette distance dans le cas de gènes signés. Cependant, le cas de gènes non-signés a été prouvé **NP-difficile** [71].

Pour deux génomes X et Y d'un même ensemble de gènes, cette distance peut être calculée à partir du graphe de points de cassure par la formule suivante [36] :

$$D(X,Y) = n - (n_c + p_0/2)$$

où n est le nombre de gènes, n_c est le nombre de cycles dans le graphe, et p_0 est le nombre de chemins pairs dans le graphe.

Plusieurs extensions de la distance DCJ on été étudiées par la suite. Par exemple, Alekseyev et al. [5, 7] ont étudié le modèle plus général *multi-breaks* qui consiste à faire de multiples coupures dans un génome. L'opération DCJ est considérée comme un cas particulier de l'opération introduite *k-break* pour k=2. Ainsi, l'opération *3-break* permet de modéliser, en plus des réarrangements standard (inversion, fission/fusion, translocation), la transposition, la transposition inversée et les 3-façons de fusion/fission. Le modèle *multi-breaks* a été utilisé pour la résolution de certaines variantes [6] du problème du *génome halving* [108]. Étant donné un génome X (unichromosomique ou multichromosomique) ayant exactement deux copies de chaque gène, ce problème consiste à trouver le nombre minimum d'opérations de réarrangement permettant de transformer X en un génome Y possédant exactement une copie de chaque gène.

Feijao et Meidanis [113] ont introduit la distance SCJ (Single Cut-or-Join) où une opération SCJ consiste à faire une coupure ou à coller deux bouts dans un génome. En se basant sur cette distance, les auteurs ont proposé des algorithmes linéaires et polynomiaux pour la résolution de certaines variantes du problème de la médiane et du génome halving. Bergeron et al. [34] ont étudié la distance SCJ_2 (*Single Cut and Join*) où une opération SCJ_2 consiste à faire une coupure et à coller deux bouts dans un génome, et ont proposé un algorithme en temps linéaire pour le calcul de cette distance.

Novac et al. [167, 169] ont étudié le modèle DCJ *restreint*, *i.e.* qui nécessite la décircularisation (ou réintégration) immédiate d'éventuel chromosome circulaire temporaire dans son chromosome original, et ont proposé un algorithme en temps $O(n \log n)$. D'autres auteurs ont considéré le modèle DCJ dans le cas de génomes ayant différents contenus en gènes (avec ou sans gènes dupliqués) en autorisant les opérations d'insertion et de délétion (DCJ-indels) [53–55, 80, 225, 254]. Ce même problème a été étudié pour le modèle DCJ restreint [51, 84]. Très récemment, Yin et al. [258] ont proposé un

algorithme de *séparation et évaluation* pour ce modèle.

Enfin, le modèle DCJ dans le cas de génomes possédant des gènes dupliqués [226, 227], et le modèle DCJ-Substitution [50, 52] qui considère l'opération DCJ et la substitution (pouvant affecter tout un chromosome) ont été considérés. Très récemment, Zeira et Shamir [260] ont introduit le modèle SCJD (SCJ/Duplication) qui considère l'opération SCJ et la duplication (pouvant affecter tout un chromosome), et ont proposé un algorithme en temps linéaire pour le calcul du nombre minimum de SCJD pour transformer un génome linéaire ordinaire (ne possédant pas de gène dupliqué) en un génome dupliqué (possédant deux copies de chaque gène) en passant uniquement par des chromosomes intermédiaires linéaires.

3.4 Modèles exemplaire, maximum et intermédiaire

Plusieurs travaux ont été proposés afin de prendre en compte les gènes dupliqués dans la comparaison de génomes [118]. Nous présentons ici les trois modèles les plus connus : le modèle de *génomes exemplaires* (*examplar genomes* en anglais) [218], le modèle maximum (*maximum matching* en anglais) [237] et le modèle *intermédiaire* [14]

3.4.1 Présentation des modèles

Dans le modèle exemplaire, Sankoff [218] pose l'hypothèse qu'un seul exemplaire (ou copie) par famille de gènes est présent dans l'ancêtre des deux génomes comparés. La stratégie suivie dans ce modèle consiste alors à trouver un **couplage** dit *exemplaire* entre les deux génomes, c'est-à-dire une correspondance entre les gènes orthologues (voir la figure 3.6(a) pour un exemple d'un couplage exemplaire). Étant donnés deux génomes X et Y possédant des gènes dupliqués, le problème de la distance exemplaire D consiste à déterminer un couplage exemplaire minimisant une certaine distance ou mesure de dissimilarité D.

Le modèle maximum, quant à lui, suppose que le maximum de gènes par famille

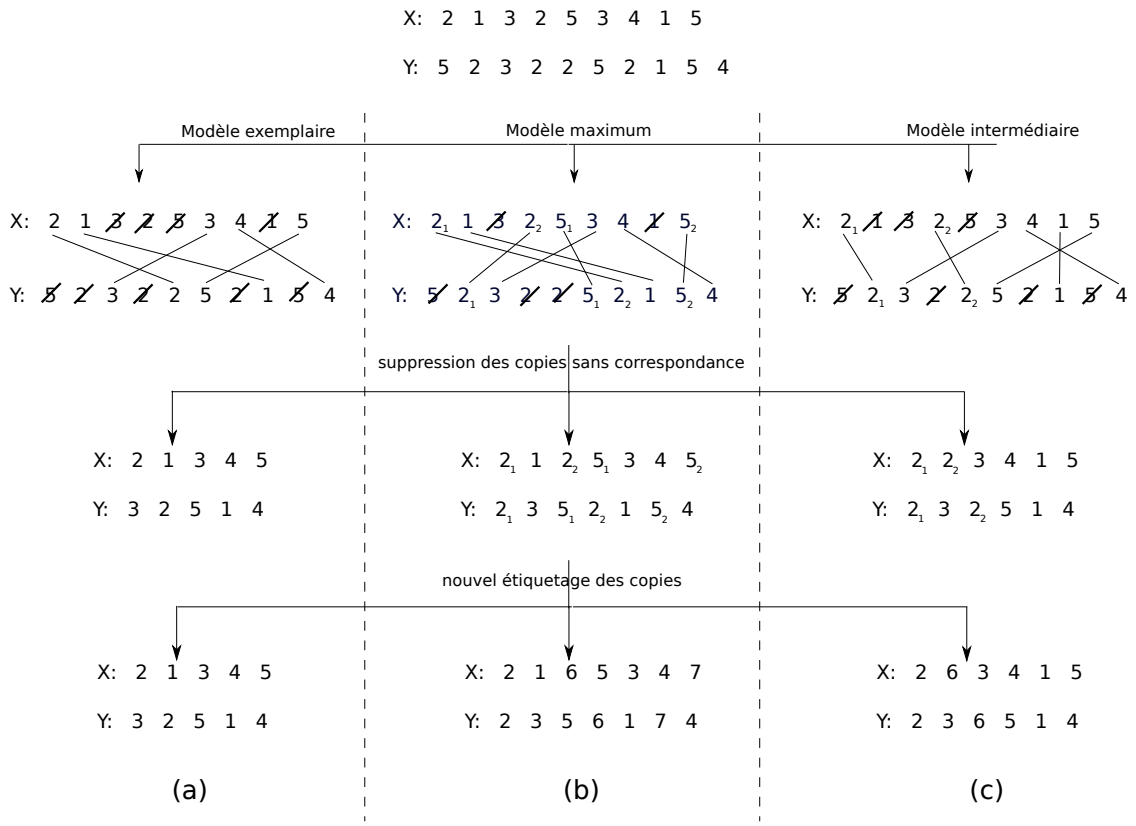


Figure 3.6 : Les différentes étapes de construction d'un couplage entre deux génomes X et Y. (a) Couplage exemplaire. (b) Couplage maximum. (c) Couplage intermédiaire.

sont présents dans l'ancêtre des deux génomes comparés. Ce modèle consiste donc à garder le maximum possible de gènes par famille dans chaque génome afin de déterminer un couplage dit *maximum* (voir la figure 3.6(b) pour un exemple d'un tel couplage). Autrement dit, si deux génomes X et Y contenant chacun un gène avec m copies et $n \geq m$ copies dans respectivement X et Y alors, le nombre de copies conservées sera m. Une fois un couplage maximum entre les génomes X et Y déterminé, les copies sans correspondance sont éliminées et les copies restantes sont étiquetées par des symboles différents pour enfin obtenir des génomes ne possédant pas de gènes dupliqués. Dans ce modèle, le choix de la copie de gène conservée dans chaque génome se base également sur un critère d'optimisation.

Enfin, le modèle intermédiaire consiste à garder au moins un gène de chaque famille. Il est considéré comme un compromis entre les deux modèles extrêmes cités précédemment. Ce modèle se base sur l'hypothèse que certains gènes sont présents en un seul exemplaire dans l'ancêtre des deux génomes, alors que d'autres apparaissent en plusieurs copies (voir la figure 3.6(c) pour un exemple d'un tel couplage).

3.4.2 Algorithmes

Le calcul de la distance exemplaire de points de cassure, et la distance exemplaire d'inversion dans le cas de gènes signés a été prouvé **NP-complet** [61].

Pour ces deux distances, Sankoff [218] a proposé un algorithme exact, de *séparation et évaluation*, en se basant sur le fait que ces deux distances vérifient la propriété de *monotonicité*, c'est-à-dire que la distance entre deux génomes n'augmente pas si l'on supprime toutes les occurrences d'une famille de gènes dans le génome. Ainsi, pour deux génomes, l'algorithme examine tous les couplages exemplaires possibles et procède comme suite. À chaque étape, ajouter une paire de gènes d'une même famille qui augmente le moins possible la distance entre les deux génomes correspondant au couplage exemplaire partiel (déjà construit), jusqu'à l'obtention d'un couplage exemplaire complet minimisant la distance entre les deux génomes.

En se basant sur les travaux de Sankoff [218], Nguyen et al. [192] ont proposé un algorithme plus rapide, de type diviser pour régner, pour le calcul de la distance exemplaire de points de cassure. Les auteurs ont précisé qu'il est possible d'étendre l'algorithme proposé pour le calcul de la distance exemplaire d'inversion.

Par la suite, certaines restrictions du problème de la distance exemplaire ont été étudiées, comme par exemple, le problème de la *distance exemplaire nulle* (*Zero-exemplar distance problem en anglais*). Étant donnés deux génomes possédant des gènes dupliqués, ce problème consiste à décider si oui ou non les deux génomes peuvent être réduits au même génome exemplaire. Cette forte restriction du problème est **NP-difficile**

quelque soit la distance utilisée [45, 154]. En revanche, le problème admet une solution polynomiale si l'on ne tient pas compte de l'ordre des gènes, avec la restriction que l'un des deux génomes possède exactement une copie de chaque gène et l'autre possède au moins une copie de chacun.

Très récemment, un algorithme exact, de programmation dynamique, a été proposé pour le calcul de la distance exemplaire de points de cassure [228].

Pour les deux modèles maximum et intermédiaire, le calcul de la distance de points de cassure et la distance d'inversion, dans le cas de gènes signés, a été prouvé **APX-difficile** [15]. Dans le cas du modèle intermédiaire, Angibaud et al. [14] ont proposé un algorithme de programmation pseudo booléenne pour la distance de points de cassure. Dans le cas du modèle maximum, Blin et al. [43] ont proposé un algorithme de type *séparation et évaluation* pour le calcul de cette distance.

D'autres distances ont été étudiées pour les deux modèles exemplaire et maximum. Pour plus de détails, nous référons le lecteur aux articles [44, 65].

3.5 Reconstruction d'ordres ancestraux

Dans cette section, nous présentons le problème de la médiane dont la solution est essentielle pour la reconstruction d'ordres ancestraux. Puis, nous citons brièvement certains algorithmes de résolution de ce problème pour la plupart des distances (points de cassure, inversion, transposition et DCJ). Nous présentons ensuite deux méthodes de reconstruction d'ordres ancestraux dans un arbre phylogénétique donné, dont nous nous servirons au chapitre 7. L'une des deux méthodes se base principalement sur le problème de la médiane. Nous citons également les algorithmes de reconstruction d'ordres ancestraux utilisant ces deux méthodes.

3.5.1 Problème de la médiane

Étant donné k génomes G_1, G_2, ..., G_k d'un même alphabet Σ, et une distance d, le problème de la médiane consiste à trouver un génome M, appelé *médian*, minimisant la somme $d(G_1,M)+d(G_2,M)+\cdots+d(G_k,M)$.

La plupart des algorithmes d'inférence de génomes ancestraux dans un arbre phylogénétique donné se basent sur la solution du problème de la médiane de trois génomes ($k=3$). C'est donc ce cas particulier que nous considérons dans la suite de ce chapitre.

Complexité

Le problème de la médiane de trois génomes a été introduit par Sankoff et Blanchette [219] pour la distance de points de cassure, et a été prouvé **NP-difficile** pour la plupart des distances (points de cassure [60, 206], inversion [68], transposition [22] et DCJ [68]), dans le cas de génomes unichromosomiques.

Dans le cas de génomes multichromosomiques et de gènes signés, ce problème a été étudié par Tannier et al. [240], et prouvé **NP-difficile** pour les deux distances d'inversion et DCJ. Les auteurs ont également prouvé que le problème est **NP-difficile** pour la distance de points de cassure dans le cas de chromosomes linéaires. En revanche, le problème admet une solution polynomiale pour la distance de points de cassure dans le cas de chromosomes circulaires ou de chromosomes mixtes (génomes possédant à la fois des chromosomes circulaires et des chromosomes linéaires).

Algorithmes et extensions

Plusieurs algorithmes exacts de type *séparation et évaluation* ont été proposés dans le but de résoudre le problème de la médiane de trois génomes pour la distance de points de cassure [219], la distance d'inversion [68, 231] et la distance DCJ [250, 252, 256, 261]. Des algorithmes d'approximation et des heuristiques ont également été proposés dans le but d'améliorer le temps de calcul pour la distance de points de cassure [62, 67, 135,

207], la distance d'inversion [47, 210, 233] et la distance DCJ [4, 176].

Certaines extensions et variantes de ce problème ont été considérées. Par exemple, Ohlebusch et al. [197] on étudié le cas des génomes de bactéries pour lesquels les réarrangements prédominants sont des inversions d'un seul gène [174], centrées autour de l'origine ou du terminus de la replication [106]. Les auteurs ont développé un algorithme linéaire pour le calcul de la médiane dans ce cas. D'autre part, Kovac et al. [169] ont répondu à une conjecture, posée par Tannier et al. [240], en prouvant que le problème de la médiane est **NP-difficile** pour le modèle DCJ restreint. Feijao et Meidanis [113] ont étudié une variante du problème de la médiane, appelée le problème de médiane pondérée, pour la distance SCJ dans le cas de génome multichromosomiques. Dans cette variante du problème, chaque génome possède un poids et la distance entre le génome médian et le génome G_i dépend du poids w_i de ce dernier. Les auteurs ont montré que ce problème admet une solution linéaire dans le cas général (*i.e.* sans restriction sur le type de chromosomes) et polynomial dans le cas de chromosomes linéaires ou circulaires. Bader [21] a étudié ce problème pour la distance d'inversion et transposition pondérée.

Récemment, le problème de la médiane pour la distance de points de cassure, dans le cas de génomes multichromosomiques et de gènes non-signés, a été étudié par Boyd et Haghighi [48]. Ces derniers ont prouvé que le problème admet une solution polynomiale dans le cas de chromosomes circulaires ou mixtes. Les auteurs ont également introduit la notion de gènes partiellement signés (*i.e.* certains gènes du génome sont signés alors que d'autres ne le sont pas) et ont prouvé que le problème de la médiane sous cette hypothèse admet une solution polynomiale pour la distance de points de cassure dans le cas de chromosomes circulaires ou mixtes. Cependant, dans le cas où le génome médian possède exactement k chromosomes, les auteurs ont prouvé que ce problème est **NP-difficile** quelque soit la nature des chromosomes (linéaires ou circulaires) et quelque soit le type des gènes des génomes (signés ou non-signés).

Très récemment, Yin et al. [257, 258] ont étudié le problème de la médiane dans

le cas de génomes ayant différents contenus en gènes, avec des gènes dupliqués. La distance considérée est DCJ avec "indels" (*i.e.* en autorisant les opérations d'insertion et de délétion de gènes), et les deux modèles étudiés sont le modèle exemplaire et le modèle maximum.

3.5.2 Méthodes et algorithmes de reconstruction d'ordres ancestraux

Nous présentons ici deux méthodes de reconstruction d'ordres ancestraux dans un arbre phylogénétique donné. Ces deux méthodes ont l'avantage de pouvoir tenir compte de n'importe quelle distance entre les génomes.

3.5.2.1 Méthode de Steinerization

C'est la méthode la plus connue pour la reconstruction d'ordres de gènes aux noeuds internes d'un arbre phylogénétique donné [40]. Cette méthode, basée sur la solution du problème de la médiane, consiste à considérer chaque noeud de l'arbre (ou ancêtre) comme la médiane des trois génomes voisins, dont l'un est son ancêtre immédiat et les deux autres ses fils. La méthode de *steinerization* se résume en trois étapes : (1) assignation initiale, aléatoire ou autre, des ordres de gènes des noeuds internes de l'arbre. (2) mise à jour des ordres des gènes, un par un, par un parcours ascendant de l'arbre en se servant, à chaque itération, des assignations les plus récentes des voisins du noeud considéré (voir Figure 3.7(a) pour un exemple). (3) répéter l'étape précédente un certain nombre de fois ou jusqu'à ce que le score de l'arbre (*i.e.* somme des distances entre les noeuds adjacents) converge vers un minimum local (plus d'amélioration des assignations).

La méthode de *steinerization* a été utilisée dans plusieurs algorithmes de reconstruction d'ordres ancestraux comme l'algorithme BPAnalysis [40] pour la distance de points de cassure, l'algorithme GRAPPA [187, 188] pour la distance de points de cassure et la distance d'inversion, l'algorithme ABC [4] pour la distance DCJ, l'algorithme

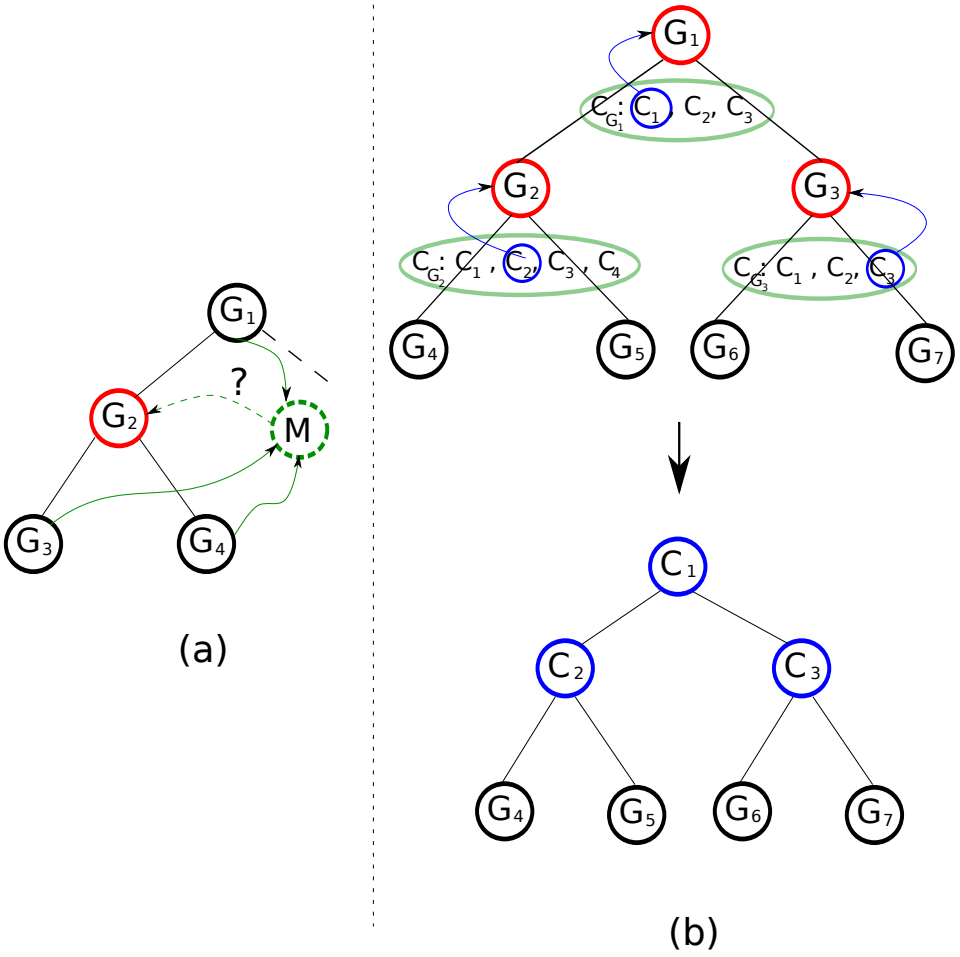


Figure 3.7 : Méthodes de reconstruction d'ordres ancestraux dans un arbre phylogénétique. (a) Une itération de la méthode de *steinerization*. Le noeud ayant comme assignation le génome G_2 sera remplacé par la médiane M s'il améliore le score de la phylogénie des noeuds G_1, G_3 et G_4. (b) Une itération de la méthode de *recherche locale*. Pour chaque noeud interne, un ensemble de candidats (délimités par un ovale) est déterminé puis, l'algorithme sélectionne le meilleur candidat (candidat en bleu).

GRAPPA-TR [259] pour la distance de transposition et l'algorithme de Bader et al. [23] pour la distance d'inversion et transposition pondérée.

3.5.2.2 Méthode de Recherche locale

Alors que la méthode de *steinerization* met à jour l'ordre des gènes d'un seul noeud de l'arbre à la fois à chaque itération, la méthode dite de *recherche locale* [168] considère tous les noeuds en même temps.

À chaque itération, un ensemble C_v de génomes candidats (ou assignations) est choisi pour chaque noeud v de l'arbre. L'ensemble de candidats peut être les génomes représentant les feuilles de l'arbre, ou des génomes intermédiaires. Plus précisément, si l'on note g_u et g_w les deux génomes des deux fils u, w de v, et d une distance donnée alors, tout génome π vérifiant $d(g_u,\pi)+d(\pi,g_w)=d(g_u,g_w)$ peut être considéré comme candidat. Les différentes médianes des génomes des trois noeuds avoisinant v, ou les génomes voisins du génome π du noeud v (*i.e.* tout génome γ vérifiant $d(\pi,\gamma)\leq 1$) peuvent être également des candidats pour le noeud v.

Une fois l'ensemble C_v des candidats de chaque noeud v de l'arbre déterminé, un score minimum $M[v,i]$ du sous arbre de racine v est calculé en deux phases, par programmation dynamique, pour chaque candidat $c_{v,i}$ de C_v. Lors de la première phase, l'arbre est parcouru de bas en haut et les différents scores des candidats des différents noeuds sont donnés par l'équation : $M[v,i]=min_j\{M[u,j]+dist(c_{v,i},c_{u,j})\}+min_k\{M[w,k]+d(c_{v,i},c_{w,k})\}$ où u et w sont les fils de v. Lors de la deuxième phase, l'arbre est parcouru de la racine aux feuilles et pour chaque noeud u de parent v ayant comme assignation le génome π, le candidat $c_{u,j}$ minimisant la somme $M[u,j]+d(\pi,c_{u,j})$ devient la nouvelle assignation du noeud u.

La figure 3.7(b) montre un exemple de sélection du meilleur candidat pour chaque noeud de l'arbre.

Si l'on note s le nombre d'espèces représentant les feuilles de l'arbre, n la taille de l'alphabet des gènes et k le nombre de candidats de chaque noeud alors le meilleur score de l'arbre, après chaque itération, peut être déterminé en temps $O(snk^2)$ dans le cas où le temps de calcul de la distance utilisée entre deux génomes est $O(n)$ (comme par exemple

le cas de la distance de points de cassure et la distance DCJ).

La méthode de *recherche locale* a été utilisée dans l'algorithme PIVO [168] et sa mise à jour PIVO2[144] (version plus rapide), qui considère la distance de points de cassure et la distance d'inversion.

D'autres algorithmes utilisent leurs propres méthodes de reconstruction d'ordres ancestraux comme MGR [47], GASTS [251] et GapAdj [124].

3.6 Conclusion

Dans ce chapitre, nous avons fait un rappel des méthodes de réarrangements existantes pour la comparaison d'ordres de gènes. Dans un premier temps, nous avons présenté les distances de réarrangements dans le cas de génomes ne possédant pas de gène dupliqué. Nous avons ensuite présenté les trois principaux modèles spécifiques à la comparaison de génomes possédant des gènes dupliqués. Enfin, nous avons présenté les deux principales méthodes d'inférence d'ordres ancestraux dans un arbre phylogénétique donné et les algorithmes qui les utilisent.

Toutefois, ces algorithmes de calcul de distances ne posent aucune contrainte quant aux espèces des génomes comparés. Or, dans le cas de comparaison de génomes d'espèces évolutivement proches, il est possible de reformuler le problème de calcul de distances en problème d'alignement de génomes. Les chapitres suivants sont consacrés à ce sujet.

CHAPITRE 4

ALIGNEMENT D'ORDRES DE GÈNES

4.1 Introduction

Dans les deux chapitres précédents, nous nous sommes intéressés à la comparaison de génomes basée sur l'alignement de leurs séquences nucléotidiques. Nous nous sommes ensuite intéressés à la comparaison d'ordres de gènes des génomes dans le but de reconstruire les ordres ancestraux des noeuds internes d'un arbre phylogénétique donné, et d'inférer des scénarios évolutifs des génomes. Dans le présent chapitre, nous faisons la connexion entre ces deux chapitres en étudiant la comparaison des contenus et ordres des gènes des génomes par une approche d'alignement.

Étant donnés deux génomes unichromosomiques X et Y, représentés par deux ordres de gènes, avec ou sans gènes dupliqués, le problème algorithmique étudié ici consiste à déterminer un ancêtre potentiel A de X et Y, minimisant le nombre d'opérations permettant de transformer A en X et A en Y. Ce problème, qui est en fait un cas particulier du problème de la petite phylogénie, est connu sous le nom du *problème de la petite phylogénie d'une cerise* (*i.e.* arbre ayant deux feuilles) ou *2 Species Small Phylogeny Problem en anglais*. Les opérations considérées dans le modèle évolutif des génomes sont les opérations modifiant le contenu en gènes (duplication, duplication inversée et perte), les réarrangements (inversion, transposition, transposition inversée) et la substitution.

Nous considérons l'hypothèse d'opérations non-chevauchantes. Plus précisément, chaque site (position dans le génome) est affecté par au plus un événement évolutif. Ces événements peuvent donc être observés (ou *visibles*) dans l'alignement des génomes. L'hypothèse d'opérations non-chevauchantes a été introduite par Schoniger et Waterman [222] dans le but de proposer une solution polynomiale au problème d'alignement de séquences nucléotidiques par inversions, étant donné que la complexité du problème

dans le cas général était inconnue. Selon les auteurs, cette hypothèse est réaliste pour la comparaison locale de séquences d'ADN évolutivement proches. En se basant sur cette hypothèse, ces auteurs ont proposé un algorithme en temps $O(n^6)$ pour le problème d'alignement de séquences nucléotidiques par inversions. Par la suite, certains travaux ont été proposés afin d'améliorer la complexité de cet algorithme, et également de proposer une solution polynomiale au problème d'alignement de séquences nucléotidiques par duplications en tandem, inversions et transpositions [75, 173, 245].

En se basant sur cette hypothèse, certains travaux ont été proposés afin d'améliorer la complexité du problème d'alignement de séquences nucléotidiques par inversions, et également de proposer une solution polynomiale au problème d'alignement de séquences nucléotidiques par duplications en tandem, inversions et transpositions [75, 173, 245].

Sous cette hypothèse, le problème de la petite phylogénie d'une cerise se ramène au problème d'alignement de deux génomes. À partir d'un tel alignement, il est possible de reconstruire l'histoire évolutive (ou scénario) des deux génomes ainsi que leur ancêtre. La restriction du problème d'alignement de deux génomes aux événements de duplication et perte a été prouvée **NP-difficile** [13]. Un premier algorithme exact de programmation linéaire pour la résolution de cette restriction a été développé dans notre laboratoire [149]. Puis, un autre algorithme de type *branch and cut* a été proposé par Andreotti et al. [13]. Ce second algorithme se base essentiellement sur l'algorithme de Holloway el al [149] avec la différence de l'ajout de certaines contraintes supplémentaires permettant de rendre ce second algorithme un peu plus rapide en pratique. Les deux algorithmes sont exponentiels dans le pire des cas mais s'avèrent en pratique efficaces pour certaines expériences avec des données biologiques (répertoires des ARN de transfert dans les génomes bactériens). Dans ce chapitre, nous présentons l'algorithme de Holloway et al. [149].

Ce chapitre est organisé comme suit : dans la section 4.2, nous définissons le mo-

dèle évolutif des génomes. Nous présentons ensuite, dans la section 4.3, le problème d'alignement de deux génomes. Enfin, nous présentons, dans la section 4.4, l'algorithme exact de programmation linéaire pour la restriction du problème d'alignement de deux génomes aux événements de duplication et perte.

4.2 Modèle évolutif de génomes

Soit $X = X_1X_2.....X_n$ une chaîne de caractères représentant un génome unichromosomique de n gènes. On désigne par $X[i]$ le i-ème gène de X, par $X[i,j]$ $(1 \leq i \leq j \leq |X|)$ le segment des gènes consécutifs $X[i]X[i+1]...X[j-1]X[j]$ de l'intervalle $[i,j]$, et par $-X[i,j] = -X[j]... - X[i+1] - X[i]$ l'inverse du segment $X[i,j]$. Soient $X[i_1,i_2]$ et $X[j_1,j_2]$ deux segments de X, tels que $i_2 < j_2$. $X[i_1,i_2]$ et $X[j_1,j_2]$ *se chevauchent* si $i_2 \geq j_1$.

- Une **Duplication** (notée D, en tandem ou transposée) de taille $(k+1)$ est une opération qui consiste à copier un segment $X[i,i+k]$ à une position j du génome X en dehors de l'intervalle $[i,i+k]$.

- Une **Perte** (notée P) de taille $(k+1)$ est une opération qui consiste à supprimer un segment $X[i,i+k]$ du génome X.

- Une **Substitution** (notée S) de taille $(k+1)$ consiste à remplacer un segment $X[i,i+k]$ par un autre segment de même taille.

- Une **Inversion** (notée I) de taille $(k+1)$ est une opération qui consiste à remplacer un segment $X[i,i+k]$ par son inverse.

- Une **Duplication Inversée** (notée DI) de taille $(k+1)$ est une opération qui consiste à copier l'inverse d'un segment $X[i,i+k]$ à une position j du génome X en dehors de l'intervalle $[i,i+k]$.

- Une **Transposition** (notée T) de taille $(k+1)$ est une opération qui consiste à supprimer un segment $X[i,i+k]$ puis l'insérer à une autre position j du génome X.

- Une **Transposition Inversée** (notée TI) de taille $(k+1)$ est une opération qui consiste à supprimer un segment $X[i,i+k]$ puis insérer son inverse à une autre position j du génome X.

On désigne par $\mathscr{O} = \{D,P,S,I,DI,T,TI\}$ l'ensemble de toutes les opérations citées précédemment. Chaque opération $O \in \mathscr{O}$ peut être représentée par deux séquences : Une *source* $O^S = X[i,i+k]$ affectée par l'opération et une *cible* $O^C = Y[j,j+k]$ résultante de l'opération. Dans le cas d'une duplication ou d'une transposition, $Y[j,j+k] = X[i,i+k]$. Dans le cas d'une inversion, d'une duplication inversée ou d'une transposition inversée, $Y[j,j+k] = -X[i,i+k]$. Et dans le cas d'une perte, $Y[j,j+k] = \emptyset$ (ensemble vide). On dénote par $O(k) = (O^S, O^C)$ une telle opération de taille k, où k est la longueur de la séquence (nombre de gènes) affectée par l'opération O, et par $c(O(k))$ le coût associé à cette opération.

Définition 19 *Soient A et X deux génomes. Une* ***histoire évolutive*** *de A vers X, notée* $O_{A\to X} = \{O_1(k_1), O_2(k_2), ..., O_l(k_l)\}$ *est une séquence d'opérations de* $\mathscr{O}$ *(possiblement vide) permettant de transformer le génome A en X. Le coût de l'histoire évolutive* $O_{A\to X}$*, noté* $c(O_{A\to X})$*, est la somme totale des coûts de ses opérations, i.e.* $c(O_{A\to X}) = \sum_{i=1}^{l} c(O_i(k_i))$.

Définition 20 *Soient maintenant deux génomes X et Y. Une* ***histoire évolutive*** *de X et Y est un triplet* $(A, O_{A\to X}, O_{A\to Y})$ *tel que A est un* ***ancêtre*** *potentiel de X et Y. Le coût de l'histoire évolutive* $(A, O_{A\to X}, O_{A\to Y})$ *est la somme des coûts* $c(O_{A\to X})$ *et* $c(O_{A\to Y})$.

Définition 21 *Soit* $(A, O_{A\to X}, O_{A\to Y})$ *une histoire évolutive de deux génomes X et Y, ayant A comme ancêtre. L'histoire* $(A, O_{A\to X}, O_{A\to Y})$ *est* ***visible*** *si pour toute opération O de* $(A, O_{A\to X}, O_{A\to Y})$*, la source et la cible sont présentes dans au moins l'un des deux*

génomes X et Y. Le génome A est, dans ce cas, appelé ***ancêtre visible*** *de X et Y. Ces opérations ont la particularité d'être non-chevauchantes, i.e. le contenu et l'ordre des gènes de la cible de chaque opération n'est pas modifié par une autre opération de réarrangement, perte ou substitution.*

Par exemple, l'histoire évolutive des deux génomes X et Y représentée sur la figure 4.1(a) est visible. En effet, la source et la cible de chaque opération sont présentes dans au moins X ou Y. Par contre, l'histoire évolutive de X et Y, représentée sur la figure 4.1(c), n'est pas visible. En effet, la source "a_1b_1" de la duplication $D = (a_1b_1, a_2b_2)$ n'est présente ni dans X ni dans Y.

Une conséquence immédiate de la définition précédente est que ces opérations sont visibles (ou observables) dans l'alignement de X et Y.

4.3 Alignement de deux génomes

On reprend ici la définition classique de l'alignement de séquences nucléotidiques.

Définition 22 *Soient X et Y deux génomes représentés par deux chaînes de caractères d'un alphabet* Σ*. Un alignement de X et Y est une paire* $(\overline{X}, \overline{Y})$ *obtenue en ajoutant à X et Y respectivement des trous (ou "gaps")* "$-$" *tel que les deux génomes alignés résultants* $\overline{X}$ *et* $\overline{Y}$ *vérifient les deux propriétés suivantes : (1)* $|\overline{X}| = |\overline{Y}|$*, (2) pour chaque position i de* $\overline{X}$ *(resp. de* $\overline{Y}$*), au plus* $\overline{X}[i]$ *ou* $\overline{Y}[i]$ *est un "gap".*

Une colonne $(\overline{X}[i], \overline{Y}[i])$ de l'alignement est appelée : (i) "gap" si $\overline{X}[i] =$ "$-$" ou $\overline{Y}[i] =$ "$-$". Dans ce cas, la position i est appelée "indel". (ii) "match" si $\overline{X}[i] = \overline{Y}[i]$. Dans ce cas, la position i est appelée "match". (iii) "mismatch" si $\overline{X}[i] \neq$ "$-$", $\overline{Y}[i] \neq$ "$-$" et $\overline{X}[i] \neq \overline{Y}[i]$. Dans ce cas, la position i est appelée "mismatch".

Les caractères de chaque colonne de l'alignement peuvent être interprétés ou associés à une opération $O \in \mathscr{O} = \{D, P, S, I, DI, T, TI\}$. Nous définissons ci-dessous un alignement interprété.

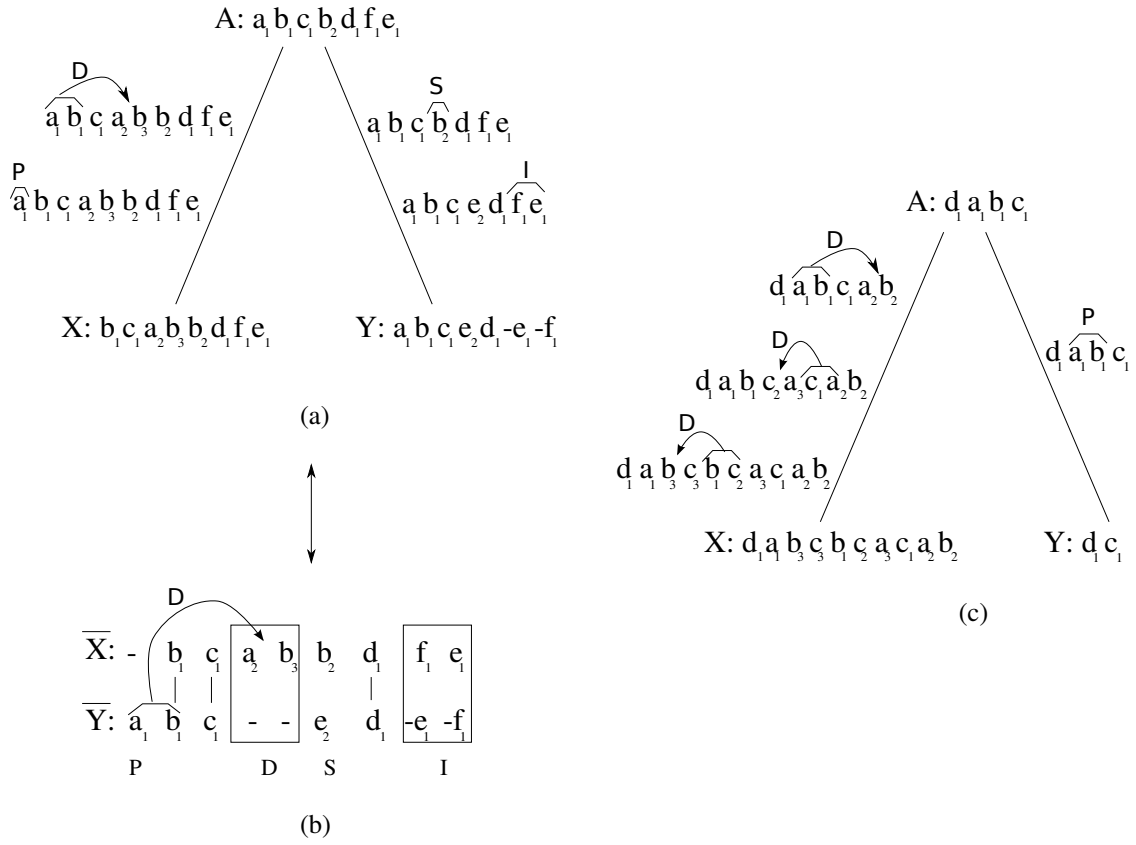


Figure 4.1 : Deux histoires évolutives de deux génomes X et Y ayant comme ancêtre le génome A, et l'alignement interprété correspondant à l'histoire visible. Les crochets indiquent les sources des différentes opérations. Le numéro au-dessous de chaque gène indique le numéro de la copie du gène. (a) Histoire évolutive visible. (b) L'alignement interprété correspondant à l'histoire évolutive montrée en (a). Un rectangle délimitant plusieurs colonnes de l'alignement signifie que les gènes appartenant à ces dernières sont tous associés à la même opération indiquée en bas. (c) Histoire évolutive non-visible. En effet, la source "a_1b_1" de la duplication $D = (a_1b_1, a_2b_2)$ n'est présente ni dans X, ni dans Y.

Définition 23 *Soit $\mathscr{A} = (\overline{X}, \overline{Y})$ un alignement de deux génomes X et Y. Une interprétation $\mathscr{L}(\mathscr{A})$ de l'alignement $\mathscr{A}$ est un ensemble d'opérations tel que : (i) chaque caractère de X (resp. de Y) est le résultat d'au plus une opération. Dans ce cas, on dit que tel caractère ou la colonne de tel caractère est associée à une opération. (ii) les caractères appartenant à une colonne "match" ne sont pas associés à une opération. (iii) le caractère appartenant à une colonne "gap" est associé à : une duplication, une duplication inversée, une transposition, une transposition inversée ou une perte. (iiii) les*

caractères appartenant à une colonne "mismatch" sont associés à une substitution ou à une inversion.

Un alignement interprété est un alignement $\mathscr{A}$ accompagné d'une interprétation $\mathscr{L}(\mathscr{A})$. Pour simplifier, on désigne par $\mathscr{L}(\mathscr{A})$ un alignement interprété.

Dans un alignement interprété, chaque opération est associée au génome contenant la cible de l'opération en question. Plus précisément, une opération est dite dans X (resp. dans Y) si la cible se trouve dans X (resp. dans Y). Dans l'histoire évolutive correspondante, une telle opération est produite sur la branche de l'arbre transformant l'ancêtre A en X (resp. en Y).

Les opérations de duplication et perte sont des opérations asymétriques, *i.e.* la source et la cible de l'opération ne peuvent pas être inter-changées. Par exemple, dans l'alignement interprété de la figure 4.1(b), la première colonne est associée à une perte du gène "a_1" dans X, la 4-ème et la 5-ème colonne sont associées à une duplication des gènes $a_1 b_1$ dans X. Une fois inférées, les duplications et pertes donnent lieu à un seul ancêtre.

Par contre, les opérations de réarrangement et de substitution sont des opérations symétriques, *i.e.* applicables indifféremment à l'un ou à l'autre des génomes (voir figure 4.2). Ces opérations nécessitent donc de fixer le génome cible, c'est-à-dire le génome dans lequel ces dernières sont produites.

Un alignement interprété $\mathscr{L}(\mathscr{A})$ a un coût $c(\mathscr{L}(\mathscr{A}))$ qui dépend des coûts des opérations associées aux caractères de ses colonnes. Par exemple, si l'on suppose que le coût de chaque opération de taille k est 1 alors l'alignement de la figure 4.1(b) a un coût de 4.

La définition précédente, à elle seule, ne permet pas de garantir une interprétation correcte de l'alignement en terme d'histoire évolutive visible. En fait, il n'est possible de retracer l'histoire évolutive visible de deux génomes X et Y, à partir de leur alignement interprété, que si ce dernier ne contient pas de cycle d'opérations (ou cycle tout court), où un cycle est défini comme suit :

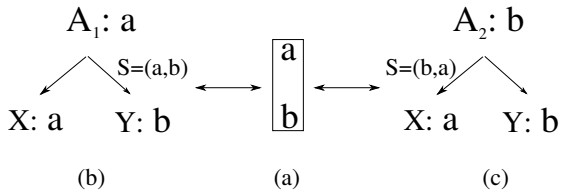


Figure 4.2 : Exemple d'un alignement interprété de deux génomes, et les deux ancêtres visibles correspondants. (a) Alignement interprété de deux génomes $X =$ "a" et $Y =$ "b". La colonne de cet alignement peut être associée à une substitution de "a" en "b" ou de "b" en "a", selon le génome où la substitution est produite. Cet alignement conduit donc à deux histoires évolutives possibles ayant deux ancêtres différents. (b) Histoire évolutive de X et Y et l'ancêtre $A_1 =$ "a" inféré dans le cas où la source de la substitution est le gène "a" de X. (c) Histoire évolutive de X et Y et l'ancêtre $A_2 =$ "b" inféré dans le cas où la source de la substitution est le gène "b" de Y.

Définition 24 *Soient $\mathscr{L}(\mathscr{A})$ un alignement interprété de deux génomes X et Y, et $\mathscr{O}_{\mathscr{A}}$ un ensemble d'opérations de $\mathscr{L}(\mathscr{A})$. $\mathscr{O}_{\mathscr{A}}$ induit un cycle s'il existe une permutation d'opérations $O_1, O_2, ..., O_k$ de $\mathscr{O}_{\mathscr{A}}$, tel que $\forall\ 1 \leq p \leq k-1$, O_p^C et O_{p+1}^S se chevauchent, et O_k^C et O_1^S se chevauchent également.*

Un alignement interprété est correct ou *réalisable* s'il ne contient pas de cycle. L'alignement interprété de la figure 4.3(c) est réalisable car il ne contient aucun cycle. L'alignement interprété de la figure 4.3(f), quant à lui, n'est pas réalisable car il contient 3 duplications $D_1 = (a_1a_2, a_3a_4)$, $D_2 = (a_3a_4, a_5a_6)$ et $D_3 = (a_5a_6, a_1a_2)$ telles que la source et la cible de chacune des trois paires $(D_1^C, D_2^S), (D_2^C, D_3^S), (D_3^C, D_1^S)$ se chevauchent. Par conséquent, il est impossible de retracer l'histoire évolutive correspondante étant donné que ces trois duplications ne peuvent pas coexister. Notez par ailleurs, qu'un cycle est constitué seulement d'opérations de duplication.

Le théorème suivant, dont la preuve est omise ici mais peut être trouvée dans [149], suppose que l'on fixe le génome cible pour les opérations de réarrangement et de substitution.

Théorème 1 *Soient X et Y deux génomes. Il y a une correspondance unique entre l'alignement interprété réalisable de X et Y, et leur ancêtre visible [149].*

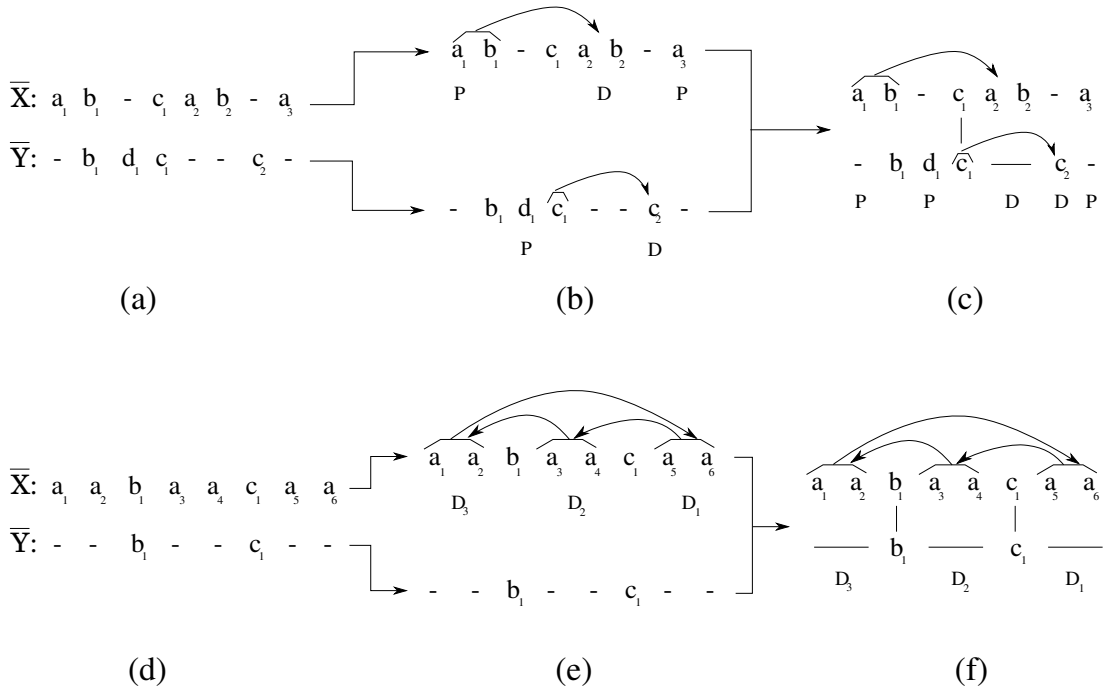


Figure 4.3 : Deux alignements de deux génomes X et Y, accompagnés respectivement de deux interprétations. (a) Alignement de X et Y. (b) Interprétation de $\overline{X}$ (resp. de $\overline{Y}$) de coût 3 (resp. 2). (c) Alignement interprété de X et Y, de coût 3+2=5, résultant des deux interprétations montrées en (b). (d) Alignement de X et Y. (e) Interprétation de $\overline{X}$ (resp. de $\overline{Y}$) de coût 3 (resp. 0). L'interprétation de $\overline{X}$ n'est pas réalisable car elle contient un cycle de trois duplications D_1, D_2 et D_3. (f) Alignement interprété de X et Y, de coût 3+0=3, résultant des deux interprétations montrées en (e). Cet alignement n'est pas réalisable car l'interprétation de $\overline{X}$ contient un cycle.

Une conséquence immédiate de ce théorème est que le problème de la petite phylogénie d'une cerise se ramène au problème suivant :

Problème d'alignement de deux génomes .

DONNÉES : Deux génomes X et Y

SORTIE : Alignement interprété réalisable $\mathscr{A} = (\overline{X}, \overline{Y})$, de coût minimum.

Une première restriction de ce problème aux opérations de duplication et perte a été tout d'abord étudiée par Holloway et al. [149]. Cette restriction, que l'on appelle par la suite *problème d'Alignement de deux génomes par Duplications et Pertes* (ou ADP par soucis de concision), a comme avantage d'inférer un seul ancêtre pour chaque aligne-

ment interprété réalisable étant donné que les duplications et pertes sont des opérations asymétriques. Afin de simplifier le problème étudié, une contrainte a été imposée : la source et la cible d'une duplication doivent appartenir au même génome. Dans ce cas, deux gènes dans X et Y respectivement ne peuvent ni appartenir à la même colonne "gap" de l'alignement, ni être associés à la même opération. Il est donc possible de traiter séparément les deux génomes alignés $\overline{X}$ et $\overline{Y}$, d'où la définition suivante :

Définition 25 Soit $\mathscr{A} = (\overline{X}, \overline{Y})$ un alignement de deux génomes X et Y sur un alphabet Σ, où $\overline{X}$ et $\overline{Y}$ sont deux génomes alignés de même taille sur l'alphabet $\Sigma^- = \Sigma \cup \{-\}$. On définit une interprétation $\mathscr{L}(\overline{X})$ du génome aligné $\overline{X}$ comme un ensemble d'opérations de duplication et perte tel que chaque position "indel" i dans $\overline{X}$, avec $\overline{X}[i] \in \Sigma$, est associée à : (i) soit une duplication $D = (X[j_1, j_2], X[i_1, i_2])$ $(1 \leq i_1 \leq i \leq i_2 \leq |\overline{X}|)$. (ii) soit une perte. Une interprétation est correcte ou réalisable si elle ne contient pas de cycle de duplications. Une interprétation $\mathscr{L}(\mathscr{A})$ de l'alignement $\mathscr{A} = (\overline{X}, \overline{Y})$ est la paire $(\mathscr{L}(\overline{X}), \mathscr{L}(\overline{Y}))$ des deux interprétations de $\overline{X}$ et $\overline{Y}$. L'union des opérations des deux interprétations de $\overline{X}$ et $\overline{Y}$ constitue l'ensemble des opérations de l'interprétation de l'alignement de X et Y. L'interprétation $\mathscr{L}(\mathscr{A})$ de l'alignement $\mathscr{A} = (\overline{X}, \overline{Y})$ est réalisable si les deux interprétations $\mathscr{L}(\overline{X})$ et $\mathscr{L}(\overline{Y})$ sont chacune réalisable (voir figure 4.3 pour un exemple).

Le coût $c(\mathscr{L}(\overline{X}))$ de l'interprétation $\mathscr{L}(\overline{X})$ est la somme des coûts de ses opérations, et le coût $c(\mathscr{L}(\mathscr{A}))$ de l'alignement interprété $\mathscr{L}(\mathscr{A})$ est la somme des coût des interprétations des deux génomes alignés.

Nous présentons maintenant le premier algorithme exact pour le problème ADP.

4.4 Algorithme exact pour le problème ADP

Nous présentons ici l'algorithme PBLP (Pseudo Boolean Linear Programming en anglais) pour trouver un alignement interprété de coût minimum en considérant uniquement les duplications et pertes. L'algorithme est basé sur la programmation linéaire à

variables booléennes (de valeur 0 et 1), et prend en entrée deux génomes X et Y de tailles respectivement n et m. L'algorithme PBLP a été implémenté en C et utilise CPLEX [1] comme solveur.

En général, un gène peut être associé à : une perte, une duplication, ou un "match" dans le cas contraire. De plus, lorsqu'un gène est associé à une duplication, cette dernière peut avoir plusieurs sources possibles. Enfin, l'ordre des colonnes "match" doit être en accord avec l'ordre des gènes dans les deux génomes X et Y. Par exemple, dans l'alignement de la figure 4.3(c), il est impossible d'avoir simultanément un "match" entre les gènes c_1 et b_2 du génome X et les gènes c_1 et b_1 du génome Y.

Nous détaillons maintenant la fonction objective, les variables et les contraintes de l'algorithme PBLP, données à la figure 4.4.

Les variables :

- La variable M^i_j représente la possibilité que le gène $X[i]$ soit un "match" avec le gène $Y[j]$.

  Comme il existe plusieurs alignements possibles entre deux génomes X et Y, il peut y avoir plusieurs possibilités de "match" entre un gène de X et les gènes de Y. On introduit pour cela les différentes variables $M^i_1, M^i_2, \cdots, M^i_{p_i}$ représentant toutes les possibilités d'avoir un "match" entre $X[i]$ et les gènes de Y.

  $M^i_j = 1$ si et seulement si $X[i]$ est un "match" avec $Y[j]$. Dans le cas contraire, $M^i_j = 0$. Chaque gène de Y possède également les mêmes variables.

- La variable DX^i_j représente la possibilité que $X[i]$ soit associé à une duplication ayant la source S_j. Il existe autant de variables que de sources de duplications correspondant au différentes cibles contenant $X[i]$.

  On désigne par $DX^i_1, DX^i_2, \ldots, DX^i_{s_i}$ les différentes variables représentant respecti-

[1] http ://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/

vement les différentes sources $S_1, S_2, \ldots, S_{s_i}$ de duplications possibles dans le cas où $X[i]$ est associé à une duplication.

Par exemple, dans le génome $X =$ "$a_1a_2b_1a_3a_4c_1a_5a_6$" de la figure 4.3(f), le premier gène $X[1] =$ "a_1" peut être associé à une duplication ayant comme source "a_2", "a_3", "a_4", "a_5" ou "a_6". De plus, comme $X[1]$ appartient au segment $X[1,2]$ qui peut être associé à une duplication ayant comme source $X[4,5]$ ou $X[7,8]$ alors il existe 7 variables DX^1_j $(1 \leq j \leq 7)$ pour $X[1]$.

$DX^i_j = 1$ si et seulement si X_i est associé à la duplication dont la source est S_j. Dans le cas contraire (*i.e.* X_i n'est pas associé à la duplication dont la source est S_j), $DX^i_j = 0$. Chaque gène de Y possède également les mêmes variables.

- La variable LX^i (resp. LY^k) représente la possibilité que $X[i]$ (resp. $Y[k]$) soit associé à une perte dans Y (resp. dans X). Il y a donc une variable pour chaque gène de X et une variable pour chaque gène de Y.

  $LX^i = 1$ (resp. $LY^k = 1$) si et seulement si $X[i]$ (resp. $Y[k]$) est associé à une perte dans Y (resp. dans X).

Les contraintes :

- Les deux premières contraintes $ct.1$ et $ct.2$ de la figure 4.4 garantissent que chaque gène des deux génomes X et Y soit associé à exactement une duplication, une perte ou un "match".

- La troisième contrainte $ct.3$ garantit que l'ordre des colonnes "match" est en accord avec l'ordre des gènes dans les deux génomes X et Y.

L'objectif de l'algorithme PBLP est de minimiser le coût de l'alignement interprété de X et Y, et donc de minimiser la somme $c_1LX^1 + \ldots + c_nLX^n + c_{n+1}LY^1 + \ldots + c_{n+m}LY^m + M^i_1 + M^i_2 + \ldots + M^i_{p_i} + c_{n+m+1}d_1 + \ldots + c_{n+m+q}d_q$ où c_l est le coût de la l-ème

Objectif : $min\ c_1 LX^1 + ... + c_n LX^n + c_{n+1} LY^1 + ... + c_{n+m} LY^m + M_1^i + M_2^i + ... + M_{p_i}^i + c_{n+m+1} d_1 + ... + c_{n+m+q} d_q$

Contraintes et domaines :

$ct1.\ LX^i + M_1^i + M_2^i + ... + M_{p_i}^i + DX_1^i + ... + DX_{s_i}^i = 1,\ \ 0 \leq i \leq n$

$ct.2\ \ LY^j + M_1^j + M_2^j + ... + M_{q_i}^j + DY_1^j + ... + DY_{r_j}^j = 1,\ \ 0 \leq j \leq m$

$ct3.\ M_j^i + M_{k1}^{l1} \leq 1, M_j^i + M_{k2}^{l2} \leq 1, ..., M_j^i + M_{k_{ti}}^{l_{ti}} \leq 1,\ \ \forall i, j$ tel que $X[i] = Y[j]$ et ($k_m \geq i$ et $l_m \leq j$), ou ($k_m \leq i$ et $l_m \geq j$).

Figure 4.4 : Fonction objective, variables et contraintes de l'algorithme PBLP.

opération, q est le nombre total des gènes de X et Y pouvant être associés à des duplications, et d_k ($1 \leq k \leq q$) est soit une variable DX_j^i du gène $X[i]$ ($1 \leq i \leq n$), ou soit une variable DY_p^r du gène $Y[r]$ ($1 \leq r \leq m$).

Notez que l'algorithme PBLP détermine la solution optimale (*i.e.* alignement interprété réalisable de coût minimum) dans le cas où cette dernière ne contient pas de cycle de duplications. Cependant, dans le cas contraire, la solution obtenue n'est pas réalisable.

Étant donné un cycle de l duplications $\{D_1, D_2, \ldots, D_l\}$ de X (ou Y, supposons que ce soit X sans perte de généralité). Afin d'empêcher la présence de tel cycle dans la solution, ces l duplications ne doivent pas coexister simultanément et donc, une ou plusieurs des l cibles de ces duplications doivent être associées à d'autres opérations. Ceci est garanti par l'ajout de la contrainte suivante à l'ensemble des contraintes initiales (voir figure 4.4) de l'algorithme PBLP.

$$d_1 + d_2 + d_3 + \cdots + d_l \leq l - 1 \tag{4.1}$$

où $d_{i_{1 \leq i \leq l}} = DX_j^i$ est la variable représentant la possibilité que les gènes de la cible D_i^C soient associés à la duplication D_i. L'algorithme PBLP est présenté ci-après (voir la page suivante).

Algorithme 1 Algorithme PBLP.

1 : ENTRÉE : Deux génomes X et Y.

2 : Déterminer une solution S (alignement interprété de X et Y)

3 : **Tant que** la solution S contient un cycle de duplication **faire**
4 : **Pour** chaque cycle trouvé $(D_1, D_2, \ldots, D_l)$ **faire**
5 : Ajouter la contrainte $d_1 + d_2 + \cdots + d_l \leq l - 1$ à l'ensemble des contraintes
6 : Déterminer une nouvelle solution S

7 : SORTIE : Retourner la solution S.

Le fonctionnement de l'algorithme est le suivant : à la ligne 2, l'algorithme détermine d'abord une solution S de coût minimum, en tenant compte des contraintes initiales seulement (voir figure 4.4). Ensuite, aux ligne 3-6, l'algorithme vérifie la présence d'éventuels cycles de duplications et ajoute les contraintes correspondantes à l'ensemble des contraintes initiales puis s'exécute à nouveau. Ce processus se répète jusqu'à ce que la solution obtenue ne contienne pas de cycle. Enfin, à la ligne 7, l'algorithme retourne l'alignement interprété des deux génomes X et Y, de coût minimum.

Notez qu'il est possible, dès le début, d'ajouter toutes les contraintes empêchant la présence d'éventuels cycles de duplications dans l'alignement obtenu. Cependant, le nombre de ces contraintes est exponentiel dans le pire des cas. En revanche, dans la pratique, il a été montré [149] que l'algorithme ne se ré-exécuter pas beaucoup de fois afin de trouver une solution optimale.

4.5 Conclusion

Ce chapitre a porté sur l'étude du problème de la petite phylogénie d'une cerise. Dans un premier temps, nous avons montré que, sous l'hypothèse d'opérations non-chevauchantes, ce problème se ramène au problème d'alignement de deux génomes. En effet, ces opérations sont observables dans l'alignement des deux génomes. Nous avons

ensuite présenté le problème d'alignement de génomes dans le cas général où l'on considère toutes les opérations possibles (duplication, perte, duplication inversée, inversion, transposition, transposition inversée et substitution). La restriction de ce problème aux opérations de duplication et perte a été prouvée **NP-difficile** [13]. Enfin, nous avons présenté l'algorithme exact PBLP de programmation linéaire pour la résolution du problème ADP d'alignement de deux génomes par duplications et pertes. Cet algorithme ainsi que l'algorithme de Andreotti et al. [13] sont exponentiels dans le pire des cas et deviennent rapidement inutilisables lorsque la taille des séquences à analyser dépasse quelques centaines de caractères. Nous étudions dans le chapitre suivant certaines variantes du problème ADP et nous proposons une heuristique rapide et efficace pour ce problème.

CHAPITRE 5

DLALIGN, HEURISTIQUE POUR LE PROBLÈME D'ALIGNEMENT DE DEUX GÉNOMES PAR DUPLICATIONS ET PERTES

5.1 Introduction

Dans le chapitre précédent, nous avons présenté le problème ADP d'alignement de deux génomes par duplications et pertes, et le premier algorithme proposé pour sa résolution. Ce problème a été prouvé **NP-difficile** [13]. Les deux algorithmes exacts de programmation linéaire existants sont exponentiels dans le pire des cas. En revanche, dans l'article de Holloway et al. [149], il a été montré que l'approche par programmation dynamique peut être utilisée pour développer une heuristique efficace pour le problème ADP. Dans ce chapitre, nous explorons en détails cette piste.

L'algorithme de programmation dynamique développé ne garantit pas que l'alignement interprété obtenu soit réalisable. En effet, ce dernier peut contenir un ou plusieurs cycles de duplications. Pour remédier à ce problème, deux solutions ont été considérées. La première est d'ignorer l'interprétation de l'alignement obtenu puis de le réinterpréter à nouveau de façon optimale, c'est-à-dire associer à nouveau les colonnes de l'alignement obtenu à un nombre minimum de duplications et pertes de sorte que l'alignement interprété résultant soit réalisable. La deuxième solution est de corriger l'alignement obtenu, en modifiant le minimum d'opérations induisant un cycle.

La première solution nous a conduit à l'étude du problème d'*Interprétation Minimale d'un Alignement* (que l'on appellera IMA par souci de concision) qui consiste à déterminer une interprétation de coût minimum d'un alignement donné de deux génomes par duplications et pertes. Nous montrons que ce problème est **APX-difficile**, même dans le cas où le nombre de copies de chaque gène ne dépasse pas 5. En revanche, le problème peut être résolu en temps linéaire dans le cas où le nombre de copies de chaque gène

est au plus 2. Il faut souligner que la preuve a été faite par notre collaborateur Riccardo Dondi avec qui nous avons publié l'article [31].

La deuxième solution conduit à l'étude du problème de *Réinterprétation Minimale d'un Alignement* (que l'on appellera RMA par souci de concision) qui consiste à corriger de façon optimale l'interprétation d'un alignement non-réalisable donné. Comme ce problème a déjà été étudié et montré **APX-difficile** [99], nous avons proposé une heuristique efficace pour sa résolution. Nous avons également développé DLAlign une heuristique par programmation dynamique, en temps quadratique, pour le problème ADP. Nous montrons que DLAlign est très rapide et fournit des résultats presque équivalents à ceux des deux algorithmes exponentiels existants.

Ce chapitre est organisé comme suit : dans la section 5.2, nous étudions la complexité du problème IMA. Nous présentons ensuite, dans la section 5.3, l'heuristique DLAlign, l'heuristique de résolution du problème RMA et les résultats de comparaison de DLAlign avec l'algorithme exact DupLoCut [13].

5.2 Étude de la complexité du problème IMA

Le problème étudié consiste à déterminer une interprétation minimale d'un alignement donné de deux génomes X et Y en considérant seulement les duplications et les pertes dans le modèle évolutif. Puisque les deux génomes alignés $\overline{X}$ et $\overline{Y}$ peuvent être considérés indépendamment l'un de l'autre, nous pouvons donc considérer uniquement un génome aligné et étudier le problème d'optimisation suivant :

Problème d'Interprétation Minimale d'un Génome Aligné (IMGA) .
DONNÉES : Un génome aligné $\overline{X}$;
SORTIE : Une interprétation réalisable $\mathscr{L}(\overline{X})$ de coût minimum.

Nous démontrons ici que le problème IMGA est **APX-difficile**, même dans le cas où le nombre de copies de chaque gène ne dépasse pas 5. En revanche, le problème peut être

résolu en temps linéaire dans le cas où le nombre de copies de chaque gènes ne dépasse pas 2. La preuve pour ce cas particulier est omise ici mais figure dans notre article [31]. Notez que dans le cas où le nombre de copies de chaque gène ne dépasse pas 4, la complexité du problème n'a pas été étudiée. Notez également que dans le cas général de coût quelconque pour les opérations, la complexité du problème reste inconnue jusqu'à présent.

Il est important de souligner que dans cette étude, nous supposons que les opérations soient de même coût. De plus, une duplication peut être de taille quelconque mais une perte peut être de taille 1 seulement. Pour simplifier, nous considérons un coût unitaire pour ces opérations dans cette étude et dans le reste de ce chapitre.

Le caractère **APX-difficile** du problème est justifié par une réduction linéaire (L-réduction) du problème CMS de couverture minimum par sommets d'un graphe cubique. Pour un bref rappel sur la L-réduction et le problème CMS, nous referons le lecteur au chapitre 1.

La preuve se résume sur les deux grandes lignes suivantes :

- Construction, en temps polynomial, du génome aligné $\overline{X}$ instance du problème IMGA, correspondant au graphe cubique G instance du problème CMS.
- Construction, en temps polynomial, de la solution de G (*i.e.* couverture par sommets de G) correspondant à la solution de $\overline{X}$ (*i.e.* interprétation réalisable de $\overline{X}$).

5.2.1 Construction du génome aligné correspondant au graphe cubique

Soit un graphe cubique $G = (V, E)$ avec $|V| = n$. Par souci de présentation, on introduit les notations suivantes : on désigne par v_i un sommet de V et par $\{v_i, v_j\}$ l'arête de E, reliant les deux sommets v_i et v_j de V. On définit ainsi un ordre sur les arêtes de E comme suit : $\{v_i, v_j\} < \{v_x, v_y\}$ si et seulement si $i < x$ ou bien $j < y$ dans le cas où $i = x$. En se basant sur cet ordre, on dénote par $\{v_1, v_a\}$ et $\{v_z, v_w\}$ respectivement la première

et la dernière arête de E, et par $\{v_i, v_{i_1}\}$, $\{v_i, v_{i_2}\}$ et $\{v_i, v_{i_3}\}$ respectivement la première, la deuxième et la troisième arête incidente à v_i ($i < i_1 < i_2 < i_3$).

Nous définissons maintenant le génome aligné $\overline{X}$ correspondant au graphe cubique G. Dans un premier temps, nous donnons une vue d'ensemble de la construction de $\overline{X}$. Toutefois, nous ignorons les positions "gap" dans le génome construit puisqu'elles ne sont pas associées à des opérations. De plus, elles n'appartiennent pas à des sources d'opérations.

Le génome aligné est composé de deux parties appelées $VE-Part$ et $A-Part$, situées respectivement à gauche et à droite dans $\overline{X}$ (voir figure 5.1). Chaque partie est composée de chaînes de caractères, chacune étant appelée *bloc*. La partie $VE-Part$ est composée de $|V|+|E|$ blocs. Pour chaque sommet v_i de V, correspond exactement un bloc $B_{VE}(v_i)$ dans $VE-Part$. De même, pour chaque arête $\{v_i, v_j\}$ de E, correspond exactement un bloc $B_{VE}(\{v_i, v_j\})$ dans $VE-part$. La partie $A-part$ est composée de $2|V|$ blocs. Pour chaque sommet v_i de V, correspond exactement deux blocs $B_{A,1}(v_i)$ et $B_{A,2}(v_i)$ dans $A-Part$.

Nous présentons maintenant en détails la structure des différents blocs du génome aligné $\overline{X}$.

Chaque position dans la partie $VE-Part$ est soit un match ou soit une "indel". Tandis que chaque position dans la partie $A-Part$ est un match, et peut appartenir à des sources de duplications associées aux chaînes de caractères des blocs de la partie $VE-Part$ qui correspondent aux sommets de V (*i.e.* les blocs $B_{VE}(v_i)$, $1 \leq i \leq n$). Par conséquent, l'ensemble de ces duplications n'induit pas un cycle. De plus, une interprétation de $\overline{X}$ est un ensemble d'opérations associées aux blocs de $VE-Part$ seulement.

Soit $\{v_i, v_j\}$ une arête de E, avec $i < j$. Par définition, $\{v_i, v_j\}$ est la p-ème arête incidente à v_i, $1 \leq p \leq 3$, et la q-ème arête incidente à v_j, $1 \leq q \leq 3$. Son bloc associé $B_{VE}(\{v_i, v_j\})$ est défini comme suit :

$$B_{VE}(\{v_i, v_j\}) = s_{e,i,j} x_{i,p} e_{i,j,1} e_{i,j,2} x_{j,q}$$

$$\overline{X} = \underbrace{B_{VE}(v_1)\dots B_{VE}(v_n)B_{VE}(\{v_1,v_a\})\dots B_{VE}(\{v_z,v_w\})}_{\text{VE-part}}\cdot$$
$$\cdot\underbrace{B_{A,1}(v_1)B_{A,2}(v_1)\dots B_{A,1}(v_n)B_{A,2}(v_n)}_{\text{A-part}}$$

Figure 5.1 : Structure du génome aligné $\overline{X}$ correspondant au graphe cubique G.

La première position du bloc $B_{VE}(\{v_i,v_j\})$ est un match, tandis que les quatre autres positions sont des "indels". Afin de pouvoir définir le bloc $B_{VE}(v_i)$, on introduit le i-encodage et le j-encodage de l'arête $\{v_i,v_j\}$ (rappelons que $i<j$). Le i-encodage de $\{v_i,v_j\}$, noté $i\text{-}enc_{i,j}$, est la chaîne de caractères définie comme suit :

$$i\text{-}enc_{i,j} = x_{i,p}e_{i,j,1}e_{i,j,2}$$

Le j-encodage de $\{v_i,v_j\}$, noté $j\text{-}enc_{i,j}$, est la chaîne de caractères définie comme suit :

$$j\text{-}enc_{i,j} = e_{i,j,1}e_{i,j,2}x_{j,q}$$

On peut maintenant définir le bloc $B_{VE}(v_i)$:

$$B_{VE}(v_i) = s_i z_{i,1}z_{i,2}\ i\text{-}enc_{i,i_1}\ z_{i,3}z_{i,4}\ i\text{-}enc_{i,i_2}\ z_{i,5}z_{i,6}\ i\text{-}enc_{i,i_3}\ z_{i,7}z_{i,8}$$

La première position du bloc $B_{VE}(v_i)$ est un match, tandis que les 17 autres positions sont toutes des "indels".

Rappelons que chaque position dans la partie $A-Part$ est un match. Ses blocs $B_{A,1}(v_i)$ et $B_{A,2}(v_i)$ sont définis comme suit.

$$B_{A,1}(v_i) = w_{i,1}z_{i,1}z_{i,2}w_{i,2}z_{i,3}z_{i,4}w_{i,3}z_{i,5}z_{i,6}w_{i,4}z_{i,7}z_{i,8}$$

$$B_{A,2}(v_i) = u_{i,1}z_{i,2}\ i\text{-}enc_{i,i_1}[1]\ u_{i,2}\ i\text{-}enc_{i,i_1}[2,3]\ z_{i,3}u_{i,3}z_{i,4}\ i\text{-}enc_{i,i_2}[1]\ u_{i,4}\ i\text{-}enc_{i,i_2}[2,3]\ z_{i,5}\cdot$$
$$\cdot u_{i,5}z_{i,6}\ i\text{-}enc_{i,i_3}[1]\ u_{i,6}\ i\text{-}enc_{i,i_3}[2,3]\ z_{i,7}$$

Avant de présenter la deuxième étape de la réduction, on définit d'abord les différentes interprétations essentielles des blocs de la partie $VE - Part$, c'est à dire l'ensemble des opérations qui peuvent leurs être associées. Rappelons qu'une interprétation de $\overline{X}$ est un ensemble d'opérations associées aux blocs de $VE - Part$ seulement, *i.e.* les blocs $B_{VE}(v_i)$ et $B_{VE}(\{v_i, v_j\})$ ($v_i \in V$).

Interprétations essentielles des différents blocs du génome aligné

La structure du génome aligné tel qu'il a été défini conduit essentiellement à deux types d'interprétations possibles pour chacun des deux blocs $B_{VE}(v_i)$ et $B_{VE}(\{v_i, v_j\})$. $B_{VE}(v_i)$ peut être associé à l'une des deux interprétations suivantes, que l'on appelle le α *labeling* et le β *labeling*.

Définition 26 Soit $G = (V, E)$ un graphe cubique, et soit $\overline{X}$ l'instance correspondante du problème IMGA. Soit v_i un sommet de V, avec $\{v_i, v_{i_1}\}$, $\{v_i, v_{i_2}\}$ et $\{v_i, v_{i_3}\}$ respectivement la première, la deuxième et la troisième arête incidente à v_i. Le α *labeling* de $B_{VE}(v_i)$ est l'interprétation constituée des opérations suivantes :

- 4 duplications ayant comme sources les quatre chaînes de caractères "$z_{i,2p-1}, z_{i,2p}$", $1 \leq p \leq 4$, de $B_{A,1}(v_i)$.
- 3 duplications ayant comme sources les trois chaînes de caractères $i\text{-}enc_{i,i_x}$, $1 \leq x \leq 3$, de $B_{VE}(\{v_i, v_{i_x}\})$.

Le β *labeling* de $B_{VE}(v_i)$) est l'interprétation constituée des opérations suivantes :

- 6 duplications ayant comme sources les six chaînes de caractères de $B_{A,2}(v_i)$ suivantes : "$z_{i,2}$ $i\text{-}enc_{i,i_1}[1]$", "$i\text{-}enc_{i,i_1}[2,3]$ $z_{i,3}$", "$z_{i,4}$ $i\text{-}enc_{i,i_2}[1]$", "$i\text{-}enc_{i,i_2}[2,3]$ $z_{i,5}$", "$z_{i,6}$ $i\text{-}enc_{i,i_3}[1]$" et "$i\text{-}enc_{i,i_3}[2,3]$ $z_{i,7}$".
- 2 pertes associées respectivement à la deuxième position ("$z_{i,1}$") et à la dernière position ("$z_{i,8}$") de $B_{VE}(v_i)$.

La figure 5.2 illustre ces deux interprétations sur un exemple d'un génome aligné correspondant au graphe complet cubique de quatre sommets $K_4 = (V, E)$. Par souci de simplicité, nous considérons uniquement le bloc $B_{VE}(v_1)$ correspondant au sommet v_1 de V.

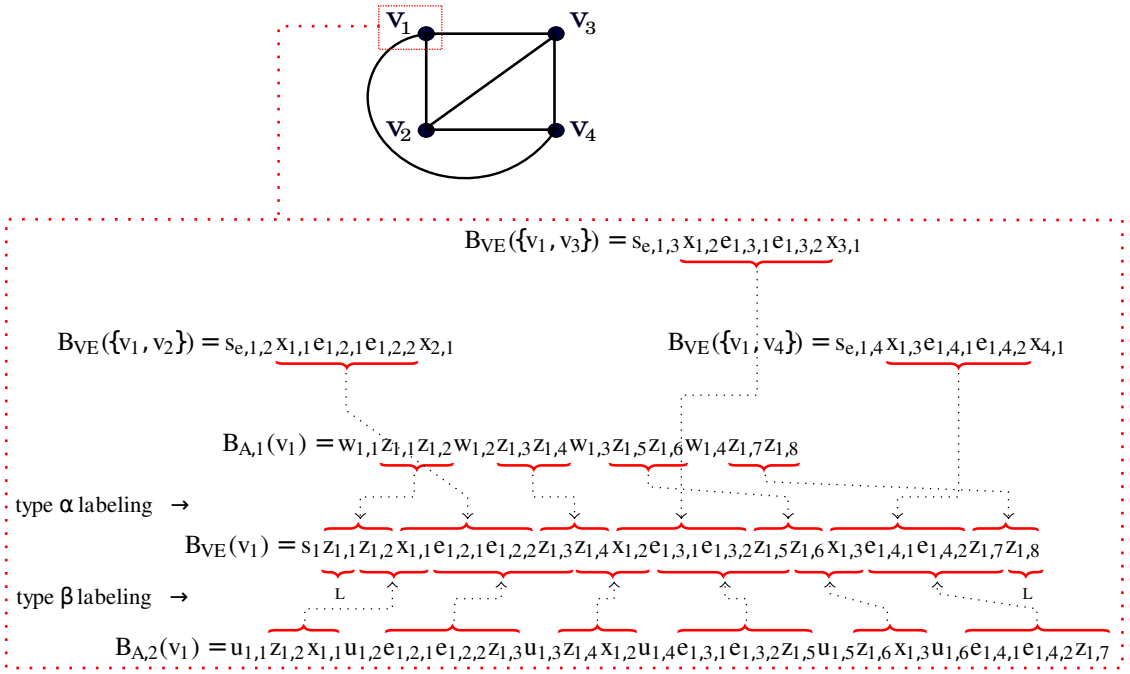


Figure 5.2 : Illustration des deux interprétations α *labeling* et β *labeling* du bloc $B_{VE}(v_1)$ du génome aligné correspondant au graphe complet cubique de quatre sommets $K_4 = (V, E)$.

D'après la définition 26, les deux interprétations α *labeling* et β *labeling* du bloc $B_{VE}(v_i)$ ont respectivement des coûts 7 et 8. De plus, le α *labeling* est optimal (*i.e.* de coût minimum, voir lemme 1). Le β *labeling* est sous-optimal (de coût 8) mais vérifie la propriété suivante :

Propriété 1 *Les opérations de l'interprétation* β labeling *de* $B_{VE}(v_i)$ *n'appartiennent pas à un cycle de duplications.*

Preuve. Comme les sources des duplications de l'interprétation β *labeling* appartiennent au bloc $B_{A,2}(v_i)$ seulement, et par construction, toutes les positions de $B_{A,2}(v_i)$ sont des "match" alors, aucune duplication peut avoir une cible dans $B_{A,2}(v_i)$. □

Définissons maintenant les deux interprétations essentielles du bloc $B_{VE}(\{v_i, v_j\})$.

Définition 27 Soit $G = (V, E)$ un graphe cubique, et soit $\{v_i, v_j\} \in E$ $(i < j)$ la p-ème arête incidente à v_i $(1 \leq p \leq 3)$ et la q-ème arête incidente à v_j $(1 \leq q \leq 3)$. Soit $\overline{X}$ l'instance correspondante du problème IMGA. Le bloc $B_{VE}(\{v_i, v_j\})$ peut être associé à l'une des deux interprétations suivantes :

- une duplication ayant comme source la chaîne de caractères "$x_{i,p}e_{i,j,1}e_{i,j,2}$" du bloc $B_{VE}(v_i)$, et une perte de la dernière position de $B_{VE}(\{v_i, v_j\})$ (caractère "$x_{j,q}$").
- une duplication ayant comme source la chaîne de caractères "$e_{i,j,1}e_{i,j,2}x_{i,p}$" du bloc $B_{VE}(v_j)$, et une perte de la deuxième position de $B_{VE}(\{v_i, v_j\})$ (caractère "$x_{i,p}$").

Ces deux interprétations ont chacune un coût de 2 et sont optimales (voir lemme 2).

Nous montrons, dans les deux lemmes suivants, que l'interprétation de type α *labeling*, et les deux interprétations de la définition 27 sont optimales.

Lemme 1 *Soient $G = (V, E)$ une instance du problème CMS et $\overline{X}$ l'instance du problème IMGA. Alors : (1) toute interprétation réalisable de $B_{VE}(v_i)$ est de coût au moins 7 ; (2) si une interprétation réalisable de $B_{VE}(v_i)$ est de coût 7, alors elle est de type α* labeling.

Esquisse de preuve. Rappelons que le bloc $B_{VE}(v_i)$ a exactement 17 positions "indels", allant de la position 2 jusqu'à la position 18. Pour démontrer (1), il suffit de remarquer que, par construction, les deux positions 2 et 18 de $B_{VE}(v_i)$ peuvent être associées respectivement à deux duplications de taille au plus 2 (voir figure 5.2). De même, les 13 positions restantes peuvent être associées à des duplications de tailles au plus 3. Donc au total, au moins $2 + \lceil \frac{13}{3} \rceil = 7$ opérations sont nécessaires et suffisantes dans une interprétation pour le bloc $B_{VE}(v_i)$.

Soit maintenant une interprétation réalisable I_{v_i} de $B_{VE}(v_i)$. Pour démontrer (2), il suffit de remarquer que dans le cas où toutes les duplications de I_{v_i} ont des sources dans

les blocs $B_{VE}\{v_i, v_j\}$, $B_{VE}\{v_i, v_h\}$, $B_{VE}\{v_i, v_k\}$ et $B_{A,1}(v_i)$ seulement alors, I_{v_i} est de coût 7 ssi I_{v_i} est de type α *labeling*. De manière similaire, si toutes les duplications de I_{v_i} ont des sources dans $B_{A,2}(v_i)$ seulement alors, I_{v_i} est de type β *labeling* et donc de coût 8. Supposons maintenant qu'il existe dans I_{v_i} une duplication D ayant une source dans $B_{A,2}(v_i)$ et une autre duplication ayant une source dans l'un des quatre blocs $B_{VE}\{v_i, v_j\}$, $B_{VE}\{v_i, v_h\}$, $B_{VE}\{v_i, v_k\}$ ou $B_{A,1}(v_i)$. I_{v_i} n'est donc ni de type α *labeling*, ni de type β *labeling*. L'idée est de montrer qu'il est possible de transformer I_{v_i} en une interprétation de type β *labeling* sans augmenter son coût. En effet, comme le coût d'une opération ne dépend pas de sa taille, D peut éventuellement être étendue à gauche et/ou à droite afin qu'elle devienne de taille maximale. Ainsi, en remplaçant toute duplication de I_{v_i}, n'ayant pas une source dans $B_{A,2}(v_i)$, par une duplication ayant une source dans $B_{A,2}(v_i)$, et en procédant de la même façon que D, I_{v_i} se transforme en un β *labeling* et donc devient de coût 8. □

Lemme 2 *Soient $G = (V, E)$ une instance du problème CMS et $\overline{X}$ l'instance du problème IMGA. Alors, toute interprétation réalisable I de $B_{VE}(\{v_i, v_j\})$, avec $\{v_i, v_j\} \in E$, est de coût au moins 2. Dans ce cas, I doit avoir une duplication ayant une source dans $B_{VE}(v_i)$ ou dans $B_{VE}(v_j)$.*

Esquisse de preuve. Par construction, le bloc $B_{VE}(v_i)$ n'a pas d'occurrence dans un autre bloc de $\overline{X}$. En tenant compte de la définition 27, il en résulte immédiatement que toute interprétation réalisable de $B_{VE}(\{v_i, v_j\})$ est de coût au moins 2.

Considérons maintenant une interprétation I n'ayant pas de duplication ni dans $B_{VE}(v_i)$ ni dans $B_{VE}(v_j)$. Alors, par construction, I est constituée d'au moins 3 opérations (2 pertes et une duplication ayant une source dans $B_{A,2}(v_i)$). □

Nous présentons maintenant la deuxième étape de la réduction.

5.2.2 Construction de la solution du graphe cubique et du génome aligné

Lemme 3 *Soient G une instance du problème CMS et $\overline{X}$ l'instance du problème IMGA. Étant donnée une couverture par sommets $V' \subseteq V$ de G, il est possible de calculer, en temps polynomial, la solution (i.e. $\mathscr{L}(\overline{X})$) du problème IMGA pour l'instance $\overline{X}$, de coût $8|V'|+7|V \setminus V'|+2|E|$.*

Esquisse de preuve. Soit V' une couverture par sommets de G. La solution ($\mathscr{L}(\overline{X})$) de l'instance $\overline{X}$ du problème IMGA est construite comme suit :

- Pour chaque sommet $v_i \in V'$, définir un β *labeling* pour le bloc correspondant $B_{VE}(v_i)$. L'interprétation de ce bloc est donc de coût 8 (voir la définition 26).
- Pour chaque sommet $v_i \in V \setminus V'$, définir un α *labeling* pour le bloc correspondant $B_{VE}(v_i)$. L'interprétation de ce bloc est donc de coût 7 (voir la définition 26).
- Pour chaque arête $\{v_i, v_j\} \in E$ (avec $i < j$), définir l'une des deux interprétations de la définition 27 pour le bloc correspondant $B_{VE}(\{v_i, v_j\})$. Précisément, c'est l'interprétation contenant une duplication ayant une source dans le bloc correspondant au sommet incident à $\{v_i, v_j\}$ qui appartient à V' (donc v_i ou v_j). L'interprétation de ce bloc est donc de coût 2 (voir la définition 27).

Comme V' est une couverture par sommets de G, au moins v_i ou v_j est dans V', il en résulte que $\mathscr{L}(\overline{X})$ est réalisable et de coût $8|V'|+7|V \setminus V'|+2|E|$. □

Lemme 4 *Soient G une instance de CMS et $\overline{X}$ l'instance du problème IMGA. Étant donnée une interprétation réalisable $\mathscr{L}(\overline{X})$ de coût $8p+7(|V|-p)+2|E|$, il est possible de calculer, en temps polynomial, la solution (i.e. couverture par sommets) de G, de taille au plus p.*

Esquisse de preuve. Étant donnée une interprétation réalisable $\mathscr{L}\overline{X}$, de coût $8p+7(|V|-p)+2|E|$. Considérons le bloc $B_{VE}(\{v_i, v_j\})$ correspondant à l'arête $\{v_i, v_j\} \in E$. L'idée

est de montrer que l'on peut supposer que l'un des deux blocs $B_{VE}(v_i)$ ou $B_{VE}(v_j)$, correspondant aux sommets incidents à $\{v_i, v_j\}$, soit associé à une interprétation de type β *labeling*. Tout d'abord, d'après le lemme 1, on peut supposer que chaque bloc $B_{VE}(v_i)$ ($v_i \in V$) est associé à une interprétation de type α *labeling* ou β *labeling*. Si $B_{VE}(v_i)$ et $B_{VE}(v_j)$ ne sont pas associés à un β *labeling* alors l'interprétation de $B_{VE}(\{v_i, v_j\})$ est de coût au moins 3 (voir lemme 2). En revanche, il est possible de déterminer, en temps polynomial, une interprétation réalisable $\mathscr{L}'(\overline{X})$ telle que $c(\mathscr{L}'(\overline{X})) \leq c(\mathscr{L}(\overline{X}))$. On procède comme suit : (1) définir un β *labeling* pour l'un des deux blocs $B_{VE}(v_i)$ ou $B_{VE}(v_j)$ correspondant à $\{v_i, v_j\} \in E$ (supposons que ce soit $B_{VE}(v_i)$) ; (2) définir l'une des deux interprétations de la définition 27 pour $B_{VE}(\{v_i, v_j\})$. Précisément, c'est l'interprétation contenant une duplication ayant comme source la chaîne de caractères $i\text{-}enc_{i,j}$ de $B_{VE}(v_i)$. Celle-ci permet de garantir de ne pas avoir d'éventuel cycle de duplications (car $B_{VE}(v_i)$ est de type β *labeling*). Par conséquent, il est possible de définir une couverture par sommets V' de G comme suit : $V' = \{v_i : B_{VE}(v_i)$ est associé à un β *labeling* dans $\mathscr{L}(\overline{X})\}$. Comme le coût de l'interprétation est $c(\mathscr{L}(\overline{X})) = 8p + 7(|V| - p) + 2|E|$, il en résulte que $|V'| \leq p$. □

Le résultat suivant est une conséquence directe des deux lemmes 3 et 4.

Théorème 2 *IMGA (et donc IMA) est* ***APX-difficile*** *même dans le cas où le nombre de copies de chaque gène dans X est au plus* 5.

Preuve. La preuve est une conséquence directe des deux lemmes 3, 4, et du fait que dans un graphe cubique G, $|E| = \frac{3}{2}|V|$ et toute couverture par sommets de G a au plus $\frac{|V|}{4}$ sommets. Ces propriétés sont nécessaires pour déterminer les deux constantes permettant de borner la solution de l'instance du problème CMS et la solution de l'instance du problème IMGA. De plus, remarquons que par construction, chaque caractère du génome aligné X a au plus 5 occurrences. □

5.3 DLAlign, heuristique pour le problème ADP

Nous présentons dans cette section, DLAlign, une heuristique en temps quadratique pour le problème d'alignement de deux génomes par duplications et pertes. L'heuristique proposée fait appel à la programmation dynamique, et se décompose en deux étapes principales. Dans un premier temps, l'heuristique détermine l'alignement interprété de coût minimum de deux génomes. Ensuite, dans le cas de présence de cycles de duplications, l'heuristique corrige l'interprétation obtenue de sorte que l'alignement interprété résultant soit réalisable et de coût aussi proche que possible du minimum. Ces deux étapes sont détaillées ci-dessous.

5.3.1 Calcul de l'alignement interprété de coût minimum

Soient X et Y deux séquences, et C une matrice de taille $(|X|).(|Y|)$ telle que chaque entrée $C[i,j]$ représente le coût minimum de l'alignement interprété des deux préfixes $X[1,i]$ et $Y[1,j]$. On considère cinq situations pour la dernière colonne de l'alignement : un match, une perte dans le génome X, une perte dans le génome Y, une duplication dans le génome X, ou une duplication dans le génome Y. Les différents coûts minimums de l'alignement interprété des préfixes $X[1,i]$ et $Y[1,j]$ pour ces différentes situations sont donnés par, respectivement, les équations $M(i,j)$, $P_X(i,j)$ et $P_Y(i,j)$, $D_X(i,j)$, $D_Y(i,j)$ ci-dessous :

- $M(i,j) = min \begin{cases} C(i-1,j-1) & \text{si } X[i] = Y[j] \\ +\infty & \text{sinon} \end{cases}$
- $P_X(i,j) = min_{0 \leq m \leq j-1}[C(i,m) + c(P(j-m))]$
- $P_Y(i,j) = min_{0 \leq m \leq i-1}[C(m,j) + c(P(i-m))]$
- $D_X(i,j) = \begin{cases} +\infty & \text{si } X[i] \text{ est un "singleton"} \\ min_{l_i-1 \leq m \leq i-1}[C(m,j) + c(D(i-m))] & \text{sinon} \qquad (*) \end{cases}$
  où $X[l_i,i]$ est le plus long suffixe de $X[1,i]$ ayant une copie dans X.

- $D_Y(i,j) = \begin{cases} +\infty & \text{si } Y[j] \text{ est un “singleton”} \\ min_{l_j-1 \leq m \leq j-1}[C(i,m) + c(D(j-m))] & \text{sinon} \end{cases}$
  où $Y[l_j, j]$ est le plus long suffixe de $Y[1,j]$ ayant une copie dans Y.

Le coût minimum $C[i,j]$ de l'alignement interprété des deux préfixes $X[1,i]$ et $Y[1,j]$ est le minimum entre les différentes valeurs de ces équations. Les initialisations pour cette heuristique sont données par :

- $C(0,0) = 0$
- $M(i,0) = M(0,j) = +\infty$
- $P_X(i,0) = P_Y(0,j) = +\infty$
- $D_X(0,j) = D_Y(i,0) = +\infty$

Comme le coût de chaque opération de taille k est 1, les équations $P_X(i,j)$ (resp. $P_Y(i,j)$) et $D_X(i,j)$ (resp. $D_Y(i,j)$) peuvent être reformulées, comme démontré dans le théorème suivant :

Théorème 3 *Les équations $P_X(i,j)$ (resp. $P_Y(i,j)$) et $D_X(i,j)$ (resp. $D_Y(i,j)$) peuvent être reformulées comme suit :*

- $P_X(i,j) = C(i,j-1) + 1$
- $P_Y(i,j) = C(i-1,j) + 1$
- $D_X(i,j) = \begin{cases} +\infty & \text{si } X[i] \text{ est un “singleton”} \\ min(C(i-1,j)+1, D_X(i-1,j)) & \text{si } D_X(i-1,j) = C(k,j)+1, \\ & \text{pour } k \geq l_i - 1 \\ C(i-1,j)+1 & \text{sinon, } \textit{i.e.} \text{ si } k < l_i - 1 \end{cases}$
  où $X[l_i, i]$ est le plus long suffixe de $X[1,i]$ ayant une copie dans X.

- $D_Y(i,j) = \begin{cases} +\infty & \text{si } Y[j] \text{ est un “singleton”} \\ min(C(i,j-1)+1, D_Y(i,j-1)) & \text{si } D_Y(i,j-1) = C(i,k)+1, \\ & \text{pour } k \geq l_j - 1 \\ C(i,j-1)+1 & \text{sinon, } \textit{i.e.} \text{ si } k < l_j - 1 \end{cases}$

  où $Y[l_j, j]$ est le plus long suffixe de $Y[1,j]$ ayant une copie dans Y.

Preuve. Nous montrons ici la preuve pour les équations $P_X(i,j)$ et $D_X(i,j)$ seulement puisque la démonstration pour $P_Y(i,j)$ et $D_Y(i,j)$ se déroule de la même manière.

Commençons par démontrer la première équation. On a $P_X(i,j) = min_{1\leq m\leq j-1}[C(i,m) + c(P(j-m))] = min_{1\leq m\leq j-1}[C(i,m)+(j-m)] = min(min_{1\leq m\leq j-2}[C(i,m)+(j-m)], C[i,j-1]+1) = min(min_{1\leq m\leq j-2}[C(i,m)+(j-1-m+1)], C[i,j-1]+1) = min(min_{1\leq m\leq j-2}[C(i,m)+j-1-m], C[i,j-1])+1$. Comme $min_{1\leq m\leq j-2}[C(i,m)+(j-1-m)] = P_X(i,j-1)$ alors $P_X(i,j) = min(P_X(i,j-1), C[i,j-1])+1$. Comme $C[i,j-1] \leq P_X(i,j-1)$, il en résulte que $P_X(i,j) = C[i,j-1]+1$.

Montrons maintenant la seconde équation. Supposons que $X[i]$ n'est pas un “singleton” et peut appartenir à la source d'une duplication D. Deux cas sont possibles : la source de D contient $X[i-1]$ ou non. Dans ce deuxième cas, clairement, $D_X(i,j) = C(i-1,j)+1$. Cette valeur est maintenue également dans le cas où $X[i-1]$ est un “singleton”.

Supposons maintenant que $X[i-1]$ n'est pas un “singleton”. Soit $[l_{i-1}, i-1]$ le plus long suffixe de $X[1,i-1]$ ayant une copie dans X. D'après la récurrence $(*)$, il existe $k \leq i-2$ tel que $D_X(i-1,j) = C(k,j)+1$ (dans le cas de plusieurs indices possibles, par exemple $k_1 < k_2 \leq i-2$, on considère le maximum ; dans cet exemple, on considère $k = k_2$). Clairement, $l_{i-1} \leq l_i$ et $l_{i-1} - 1 \leq k \leq i-2$. Notez que l'indice k de la récurrence $D_X(i,j)$ est égal soit à l'indice k de la récurrence $D_X(i-1,j)$ ou soit à $i-1$.

On a $D_X(i,j) = min_{l_i-1\leq m\leq i-1}[C(m,j) + c(D(i-m))] = min_{l_i-1\leq m\leq i-1}[C(m,j)+1] = min(min_{l_i-1\leq m\leq i-2}[C(m,j)+1], C(i-1,j)+1)$.
Pour simplifier, on pose $Z = min_{l_i-1\leq m\leq i-2}[C(m,j)+1]$. Donc, $D_X(i,j) = min(Z, C(i-1,j)+1)$. On distingue deux cas en fonction de la valeur de k :

Cas 1 : $k \geq l_i - 1$. Comme $D_X(i-1,j) = min_{l_{i-1}-1 \leq m \leq i-2}[C(m,j)+1]$ et $l_{i-1} \leq l_i$, alors $min_{l_{i-1}-1 \leq m \leq i-2}[C(m,j)+1] = min_{l_i-1 \leq m \leq i-2}[C(m,j)+1] = Z$ (voir figure 5.3), et donc

$$D_X(i,j) = min(Z, C(i-1,j)+1) = min(D_X(i-1,j), C(i-1,j)+1) \tag{5.1}$$

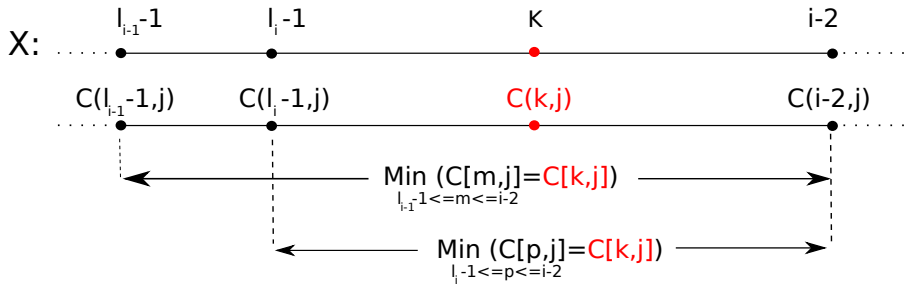


Figure 5.3 : Schéma expliquant la preuve du premier cas de la récurrence $D_X(i,j)$. Comme $min_{l_{i-1}-1 \leq m \leq i-2}[C(m,j)] = C(k,j)$ et $k \geq l_i - 1 \geq l_{i-1} - 1$ alors, $min_{l_i-1 \leq m \leq i-2}[C(m,j)] = C(k,j)$.

Cas 2 : $k < l_i - 1$. Alors, $min_{l_i-1 \leq m \leq i-2} C(m,j) > C(k,j)$ (rappelons que k est le maximum parmi tous les indices possibles). De plus, $Z = min_{l_i-1 \leq m \leq i-2}[C(m,j)+1] > C(k,j)+1 = D_X(i-1,j)$. Donc, $Z \geq D_X(i-1,j)+1 \geq C(i-1,j)+1$ (puisque $C(i-1,j)$ est le coût minimum de l'alignement interprété des préfixes $X[1,i-1]$ et $Y[1,j]$), et donc

$$D_X(i,j) = min(Z, C(i-1,j)+1) = C(i-1,j)+1 \tag{5.2}$$

□

Remarquons que chaque valeur des équations de récurrence $M(i,j)$, $L_X(i,j)$ et $L_Y(i,j)$ peut être calculée en temps constant. Pour les deux équations $D_X(i,j)$ et $D_Y(i,j)$, les différents indices des différents préfixes considérés de X et Y peuvent être stockés dans un

tableau et ainsi retrouvés en temps constant. De plus, en utilisant l'arbre des suffixes, qui peut être construit en temps linéaire, les plus longs suffixes des préfixes de X (resp. de Y), ayant une copie dans X (resp. dans Y), peuvent être retrouvés en temps constant. Par conséquent, chaque valeur des équations $D_X(i,j)$ et $D_Y(i,j)$ peut également être calculée en temps constant.

Les équations de récurrence précédentes permettent de calculer le coût minimum de l'alignement interprété des deux génomes X et Y. Pour trouver le ou les alignements interprétés correspondants, il faut parcourir la matrice entière en sens inverse. De plus, il est nécessaire de stocker, pour chaque entrer $C(i,j)$ de la matrice, les différentes positions des sources des opérations qui peuvent être associées aux gènes $X[i]$ et $Y[j]$.

Notez que lors du parcours de la matrice en sens inverse, chaque chemin emprunté permet de construire un alignement interprété différent, et donc donner lieu à une histoire évolutive et un ancêtre différents.

Cependant, l'alignement interprété déterminé n'est pas garanti réalisable et peut donc avoir un ou plusieurs cycles de duplications. Par conséquent, l'interprétation de l'alignement nécessite une correction. Afin de limiter la possibilité d'avoir un tel cycle dans l'alignement, on procède comme suit : Soit $T = X[i,\ldots,i+k]$ un segment de X qui peut être associé à une duplication ayant plusieurs sources possibles (*i.e.* T a plusieurs copies dans X). Alors, on choisit la copie la plus à gauche dans X comme source de duplication pour T.

L'intérêt de cette méthode est de permettre, après la seconde étape de l'heuristique, d'obtenir un alignement interprété réalisable de coût aussi proche que possible de l'optimal. En effet, à titre d'exemple, les deux alignements interprétés non-réalisables des deux génomes $X =$ "$abcabdababeabfab$" et $Y =$ "$cdef$", montrés en figure 5.4, contient chacun 6 duplications, avec la différence que l'alignement montré en (a) compte 3 cycles de duplications alors que celui montré en (b) en compte seulement 1. La correction de l'interprétation non-réalisable de l'alignement montré en (a) conduit à l'alignement in-

terprété réalisable, de coût 9. Tandis que la correction de l'interprétation non-réalisable de l'alignement montré en (b) conduit à l'alignement interprété réalisable, de coût 7. Ce dernier coût est en fait le coût minimum de l'alignement interprété réalisable de X et Y.

5.3.2 Correction de l'interprétation de l'alignement non-réalisable

Étant donné un alignement interprété non-réalisable $\mathscr{L}(\overline{X},\overline{Y})$ de deux génomes X et Y, obtenu lors de la première étape de l'heuristique, l'objectif est de corriger, s'il y a lieu, l'interprétation de l'alignement en associant à certains gènes de X et Y de nouvelles opérations de sorte que l'alignement interprété résultant soit réalisable, et donc ne contienne pas de cycles. Avant de décrire précisément le problème considéré, nous donnons certaines définitions nécessaires.

Rappelons que les deux génomes alignés $\overline{X}$ et $\overline{Y}$ peuvent être traités séparément, et donc l'interprétation $\mathscr{L}(\overline{X},\overline{Y})$ peut être décomposée en deux interprétations $\mathscr{L}(\overline{X})$ et $\mathscr{L}(\overline{X})$. Supposons qu'au moins une des deux interprétations soit non-réalisable. Sans perte de généralité, supposons que ce soit $\mathscr{L}(\overline{X})$. Tout d'abord, on définit une *réinterprétation* $\mathscr{L}'(\overline{X})$ de $\mathscr{L}(\overline{X})$ comme une interprétation de $\overline{X}$ obtenue en remplaçant l'ensemble $(D_1,\cdots,D_p)$ des duplications de $\mathscr{L}(\overline{X})$, induisant des cycles, par des pertes. Le coût de la réinterprétation $\mathscr{L}'(\overline{X})$ est donné par l'équation suivante :

$$c(\mathscr{L}'(\overline{X})) = c(\mathscr{L}(\overline{X})) + \sum_{1\leq i\leq p} (|D_i|-1) \tag{5.3}$$

où $|D_i|$ est la taille de la duplication D_i. On définit ainsi le *coût de réinterprétation*, que l'on désigne par $c_r(\mathscr{L}'(\overline{X}))$, de $\mathscr{L}'(\overline{X})$ par rapport à $\mathscr{L}(\overline{X})$ comme suit :

$$c_r(\mathscr{L}'(\overline{X})) = c(\mathscr{L}'(\overline{X})) - c(\mathscr{L}(\overline{X})) = \sum_{1\leq i\leq p} (|D_i|-1) \tag{5.4}$$

Le problème d'optimisation considéré est donc le suivant :

Problème de Réinterprétation Minimale d'un Alignement (RMA) .
DONNÉES : Une interprétation non-réalisable $\mathscr{L}(\overline{X})$ d'un génome aligné $\overline{X}$;

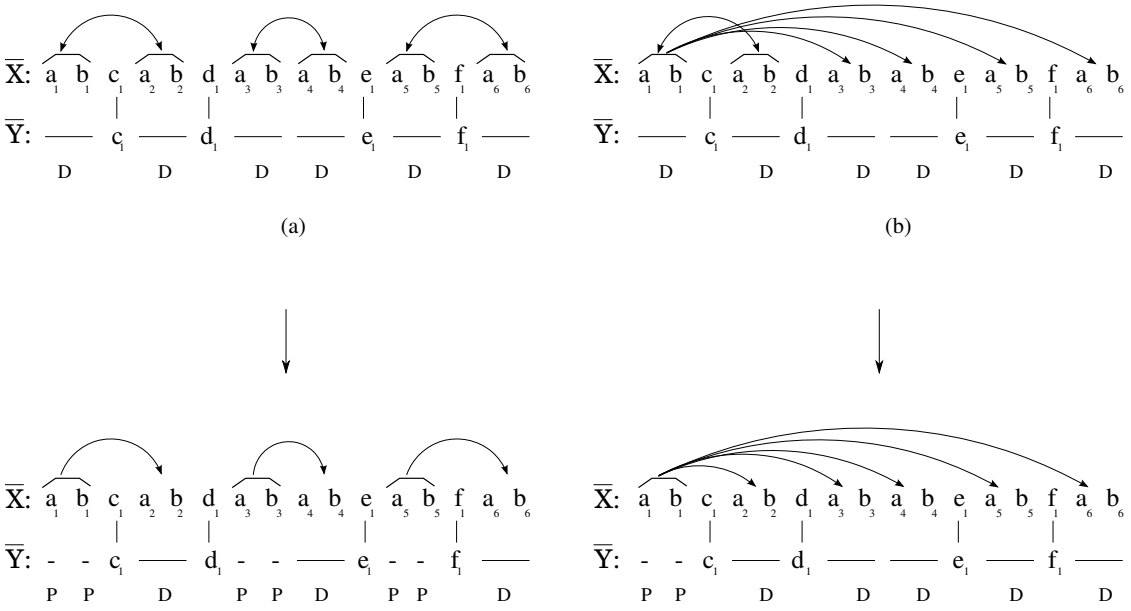


Figure 5.4 : Deux alignements non-réalisables possibles de même coût, de deux génomes X et Y. (a) Alignement non-réalisable conduisant à un alignement réalisable de coût 9. (b) Alignement non-réalisable conduisant à un alignement réalisable de coût 7.

SORTIE : Une réinterprétation réalisable $\mathscr{L}'(\overline{X})$ de $\mathscr{L}(\overline{X})$ de coût de réinterprétation minimum.

Comme ce problème a été étudié et prouvé **APX-difficile** [99], nous proposons ici une heuristique pour sa résolution. Tout d'abord, on associe à l'ensemble $D = \{D_1, \cdots, D_{k \leq n}\}$ de toutes les duplications de $\mathscr{L}(\overline{X})$ un graphe orienté que l'on appelle *Graphe de duplications* défini comme suit :

Définition 28 *Un graphe de duplications associé à un ensemble de duplications* $D = \{D_1, \cdots, D_{k \leq n}\}$ *est un graphe* $G = (V, E)$ *orienté et pondéré, construit comme suit :*

Pour toute duplication D_i *de D, on associe deux sommets* s_i *et* c_i*, correspondant respectivement à la source* s_i *et la cible* c_i *de* D_i*, et une arête orientée* (s_i, c_i).

Pour toute paire (s_i, c_j) $(i \neq j)$ *de source et cible qui se chevauchent, on associe une arête orientée* (s_i, c_j)*, correspondant au chevauchement* $(s_i \cap c_j)$*, ayant comme poids* $|(s_i \cap c_j)|$.

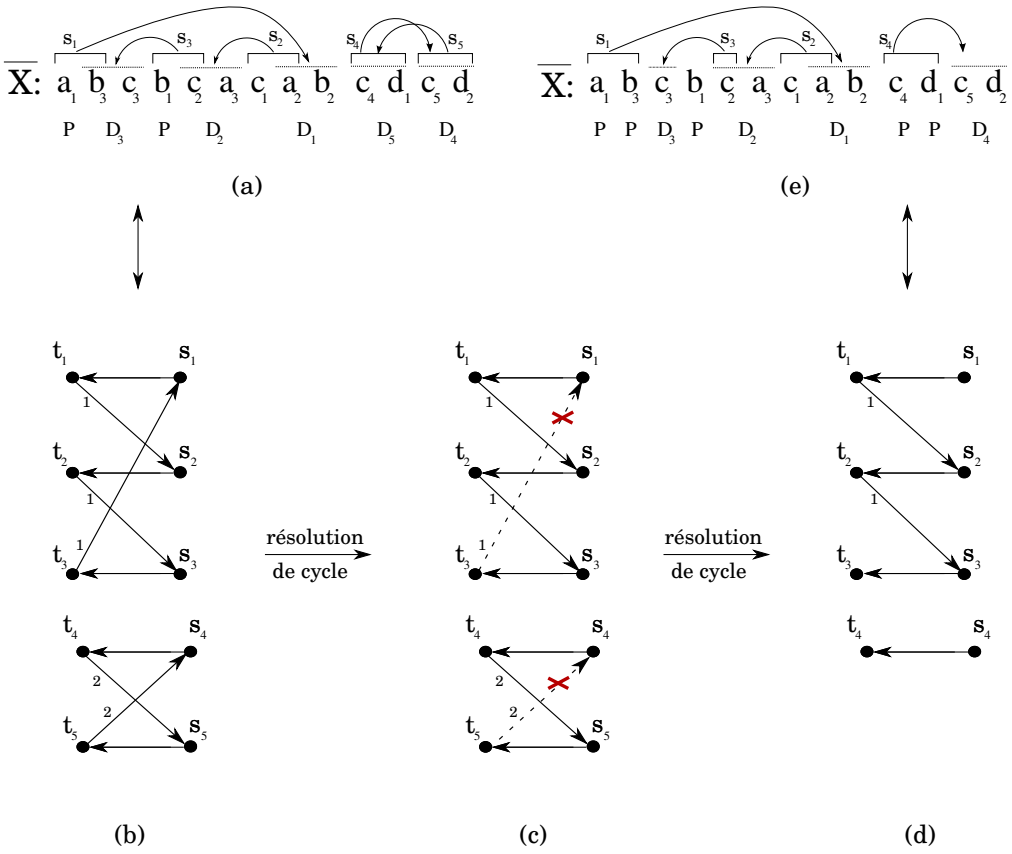


Figure 5.5 : Les différentes étapes de résolution du problème RMA par l'heuristique 2, pour le génome $X =$ "*abcbcacabcdcd*". (a) Le génome X aligné, accompagné d'une interprétation $\mathscr{L}(\overline{X})$ non-réalisable. Un crochet et une barre horizontale pointillée indiquent respectivement la source et la cible de chaque duplication. (b) Le graphe de duplications $G = (V, E)$ associé à l'ensemble $(D_1, D_2, D_3, D_4, D_5)$ des duplications de $\mathscr{L}(\overline{X})$. (c) Détection et élimination des cycles dans le graphe G. Lors du premier parcours de G, le cycle $\mathscr{B}_1 = (s_1, t_1, s_2, t_2, s_3, t_3, s_1)$ est détecté puis l'arête (t_3, s_1) de poids minimum est supprimée. Ce choix est arbitraire dans ce cas puisque les trois arêtes sont de poids minimum. La suppression de cette arête conduit à l'association du gène b_3 à une perte. Lors du deuxième parcours du graphe, le cycle $\mathscr{B}_2 = (s_4, t_4, s_5, t_5, s_4)$ est détecté puis l'arête (t_5, s_4) de poids minimum est supprimée. Ceci conduit à l'association des gènes "$c_4 d_1$" à des pertes, et à la suppression des deux arêtes (s_5, t_5) et (t_4, s_5) puisque la duplication D_5 n'existe plus. (d) Le graphe acyclique résultant après l'étape (c). (e) L'alignement interprété réalisable, obtenu après l'élimination des cycles du graphe G.

La figure 5.5(a,b) donne un exemple d'un alignement interprété non-réalisable et le graphe de duplications associé à l'ensemble de ses duplications.

Remarquons que les deux types de sommets s_i et c_i $(1 \leq i \leq k)$ du graphe de duplications sont respectivement de degré sortant 1 et de degré entrant 1. On peut aussi remarquer que certaines arêtes du graphe ne sont pas pondérées, ces arêtes correspondent

en fait à des paires de source et cible appartenant à la même duplication.

À partir de la définition du graphe de duplications, le problème RMA se ramène au problème de suppression d'un minimum d'arêtes pondérées de E, ayant une somme de poids minimale, tel que le graphe de duplications résultant soit **acyclique**. Cette reformulation du problème est similaire au problème connu sous le nom de *Feedback Arc Set problem* (FAS) [119], qui consiste à déterminer, dans un graphe donné, le plus petit ensemble d'arêtes de poids minimum tel que le graphe résultant soit acyclique. A la différence du problème FAS, dans notre reformulation du problème RMA, l'ensemble des arêtes candidates est restreint aux arêtes pondérées.

Le problème FAS est **NP-Difficile** dans le cas général [127, 158]. Des heuristiques ont été proposées [91, 103, 104, 122, 146, 177], ainsi que des solutions polynomiales pour certaines classes de graphes comme les graphes planaires orientés [184], et les graphes de flot reductibles [247]. Pour plus de détails sur les algorithmes de résolution du problème FAS, on renvoie le lecteur à l'état de l'art réalisé dans [119].

Notez que l'ensemble minimum d'arêtes à supprimer dans un graphe, afin de le rendre acyclique, n'est pas unique. Ainsi, une interprétation non-réalisable peut avoir plusieurs réinterprétations réalisables de coût de réinterprétation minimum. Notez également que la difficulté du problèm FAS repose sur le fait que dans un graphe quelconque, les cycles peuvent partager des arêtes. Dans le cas contraire, le problème devient beaucoup plus simple puisqu'il suffit de supprimer une arête de chaque cycle. C'est le cas de la plupart des graphes de duplications obtenus à partir des alignements de données biologiques réelles et de données simulées qui sont composés de cycles ne partageant pas d'arêtes pondérées. Par conséquent, l'heuristique proposée ci-dessous permet de résoudre efficacement le problème RMA, et ainsi d'obtenir une réinterprétation réalisable de coût de réinterprétation minimum dans la plupart des cas.

L'idée qui est exploitée par l'heuristique est assez simple. À partir d'un génome aligné $\overline{X}$ et une interprétation $\mathscr{L}(\overline{X})$, on calcul d'abord le graphe de duplications $G = (V, E)$

Algorithme 2 Heuristique pour le problème RMA.

1 : ENTRÉE : Un génome aligné $\overline{X}$ et une interprétation non-réalisable $\mathscr{L}(\overline{X})$;

2 : Construire le graphe de duplications $G = (V, E)$ associé à l'ensemble $D = \{D_1, \cdots, D_{k \leq n}\}$ des duplications de $\mathscr{L}(\overline{X})$

3 : Tant que G est cyclique faire :
4 : soit $\mathscr{B}$ un cycle de G et (c_i, s_j) une arête de C de poids minimum
5 : - supprimer (c_i, s_j) et associer aux gènes du segment $(c_i \cap s_j)$ de $\mathscr{L}(\overline{X})$ des pertes.
6 : - mettre à jour la taille de la duplication D_i, ainsi que le graphe G

7 : SORTIE : Une réinterprétation réalisable $\mathscr{L}'(\overline{X})$;

associé à l'ensemble $D = \{D_1, \cdots, D_{k \leq n}\}$ de toutes les duplications de $\mathscr{L}(\overline{X})$. Ensuite, on parcourt le graphe G plusieurs fois et à chaque parcours, on cherche d'éventuels cycles. Si un cycle $\mathscr{B}$ est détecté alors, on supprime l'arête (c_i, s_j) de $\mathscr{B}$ de poids minimum et l'on associe aux gènes du segment $(c_i \cap s_j)$ des pertes.

Comme ces gènes ont été auparavant associés à la duplication D_i, il est nécessaire de mettre à jour la taille de cette dernière, c'est à dire, modifier la taille de la cible s_i et de la source c_i, et par conséquent, mettre à jour le poids de l'arête (c_i, s_j), ainsi que les différents poids des arêtes (c_p, s_i) et (c_i, s_p) $(1 \leq p \leq k)$ dans le cas où les segments c_p et s_i et/ou les segments c_i et s_p se chevauchent toujours. Cette étape pourrait conduire à la suppression de l'arête (s_i, c_i) si la duplication D_i n'existe plus, et à la suppression des arêtes (c_p, s_i) et/ou (c_i, s_p) $(1 \leq p \leq k)$ dans le cas où les segments c_p et s_i et/ou les segments s_p et c_i ne se chevauchent plus. Le graphe est alors mis à jour en tenant compte de ces modifications.

Les étapes précédentes sont effectuées jusqu'à ce que le graphe devienne acyclique (voir une illustration à la figure 5.5).

5.3.3 Analyse de complexité

On analyse dans cette section la complexité de l'heuristique DLAlign. Pour simplifier, on pose $|X| = |Y| = n$.

La première étape de l'heuristique se fait en temps $O(n^2)$. En effet, chaque valeur des équations de récurrence se calcule en temps constant. Le calcul de chaque entrée de la matrice C se fait donc en temps constant, et par conséquent, la complexité pour déterminer la matrice entière est de l'ordre de $O(n^2)$. La construction de chaque alignement interprété possible nécessite un "backtracking" dans la matrice, qui se fait en temps $O(n)$.

Dans la deuxième étape de l'heuristique, la construction du graphe de duplications peut se faire en temps $O(n^2)$. La détection d'un cycle $\mathscr{B}$ et la suppression de l'arête (c_i, s_j) de $\mathscr{B}$ de poids minimum peut se faire en temps $O(n)$, par un algorithme de parcours en profondeur d'un graphe [81]. La mise à jour du graphe G nécessite la modification des poids des arêtes incidentes à c_i et à s_j. Comme le nombre de ces arêtes est $O(n)$, la mise à jour du graphe se fait donc en temps $O(n)$.

Enfin, comme le nombre d'arêtes du graphe G est au plus $O(n)$ (voir théorème 4), on en déduit que le temps nécessaire pour rendre le graphe G acyclique est $O(n^2)$. La deuxième étape de l'heuristique se fait donc en temps $O(n^2)$, qui est également la complexité de l'heuristique.

Notez par ailleurs que dans le cas de coût quelconque pour les duplications et pertes, le calcul des valeurs des équations de récurrence se fait en temps linéaire et donc la complexité de l'heuristique sera de l'ordre de $O(n^3)$.

Théorème 4 *Le graphe de duplications $G = (V,E)$, associé à l'ensemble $(D_1, \cdots, D_k)$ des duplications d'une interprétation $\mathscr{L}(\overline{X})$ donnée, a au plus $O(n)$ arêtes, où $n = |X|$.*

Preuve. On désigne par $d^+(v)$ l'ensemble des arêtes sortantes du sommet v, et par G_x l'ensemble des gènes du segment x du génome X. Rappelons que k est le nombre de

duplications de $\mathscr{L}(\overline{X})$. Soit C_i l'ensemble de toutes les sources de duplications qui se chevauchent avec la cible c_i. Formellement, $C_i = \{s_j : G_{s_j} \cap G_{t_i} \neq \emptyset\}$. D'après la définition du graphe de duplications, on a $d^+(s_i) = 1$ et $d^+(t_i) = |C_i|$, pour tout i, et donc $|E| = \sum_{i=1}^{k} d^+(s_i) + \sum_{i=1}^{k} d^+(t_i) = k + \sum_{i=1}^{k} |C_i|$. Il reste à montrer que $\sum_{i=1}^{k} |C_i| \leq n$.

On pose $C_i = \{s_{l_1}, \cdots s_{l_p}\}$ et $\gamma_i = \sum_{j=1}^{p} |G_{t_i} \cap G_{s_{l_j}}|$. Il est clair que $\sum_{i=1}^{k} |G_{t_i} \cap G_{s_j}| \leq |G_{s_j}|$, donc

$$\sum_{j=1}^{k} (\sum_{i=1}^{k} |G_{t_i} \cap G_{s_j}|) \leq \sum_{j=1}^{k} |G_{s_j}|$$

Comme $\sum_{j=1}^{k} (\sum_{i=1}^{k} |G_{t_i} \cap G_{s_j}|) = \sum_{i=1}^{k} (\sum_{j=1}^{k} |G_{t_i} \cap G_{s_j}|) \geq \sum_{i=1}^{k} \gamma_i$ et $\sum_{i=1}^{k} |G_{s_i}| = \sum_{i=1}^{k} |G_{t_i}|$ (puisque la source et la cible d'une même duplication sont de même taille), on a donc

$$\sum_{i=1}^{k} \gamma_i \leq \sum_{i=1}^{k} |G_{t_i}| \tag{5.5}$$

De plus, comme $|G_{t_i} \cap G_{s_{l_j}}| \geq 1$ pour tout $1 \leq j \leq p$ (d'après la définition de C_i), on a $\gamma_i \geq p \geq |C_i|$, et donc

$$\sum_{i=1}^{k} |C_i| \leq \sum_{i=1}^{k} \gamma_i \tag{5.6}$$

Enfin, comme $G_{t_i} \cap G_{t_j} = \emptyset$ pour tout i, j tel que $i \neq j$, on a donc

$$\sum_{i=1}^{k} |G_{t_i}| \leq n \tag{5.7}$$

À partir des inégalités 5.5-5.7, on conclut que $\sum_{i=1}^{k} |C_i| \leq n$, achevant la preuve. □

5.3.4 Résultats expérimentaux

Pour mesurer la performance de l'heuristique DLAlign, nous avons effectué des expériences portant sur la comparaison de DLAlign avec l'algorithme exact de programmation linéaire DupLoCut. L'heuristique DLAlign a été implémentée en Java, et les

expériences ont été réalisées sur une machine dotée d'un processeur 8-core Intel(R) 3.6 GHZ avec 16GB de RAM. Le coût considéré pour une duplication ou une perte est 1. La duplication peut être de taille quelconque alors qu'une perte est de taille 1. Notez qu'il est possible de choisir un coût quelconque pour ces opérations à l'exception, pour des exigences méthodologiques, d'un coût de 1 pour une perte de taille quelconque. En effet, tout alignement interprété est, dans ce cas, de coût 2 puisqu'il suffit d'associer tous les gènes de chaque génome à une seule perte. Pour éviter les biais induits par ce choix de coût, une étape de post-traitement est effectuée, durant laquelle tous les gènes consécutifs associés à plusieurs pertes seront associés à une seule perte.

Notre heuristique a été testée sur des simulations. Chaque jeu de données comporte plusieurs simulations de couples de séquences (X,Y) générées de la façon suivante : Une séquence aléatoire R de taille n sur un alphabet de taille α est d'abord générée ensuite, l opérations (duplications et pertes non-chevauchantes) sont effectuées sur R pour obtenir une séquence ancestrale A. Enfin, deux séries de l opérations sont effectuées sur A pour obtenir respectivement les deux séquences X et Y. La taille de la duplication suit une distribution gaussienne de moyenne 5 et d'écart type 2 dans le cas de l'évolution des répertoires d'ARN de transfert des Bacillus [149], dont nous parlerons au chapitre suivant.

5.3.4.1 Temps d'exécution

Dans cette expérience, nous comparons les temps d'exécution de l'heuristique DLAlign et de l'algorithme exact DupLoCut sur plusieurs jeux de données.

Notez que la taille de l'alphabet α n'a pas d'influence sur le temps de calcul de l'heuristique DLAlign. Cependant, dans le cas de l'algorithme DupLoCut, plus cette valeur diminue, plus le temps de calcul devient important. Nous avons donc choisi de fixer ce paramètre à une valeur moyenne $\alpha = \frac{n}{10}$ pour tous les jeux de données.

Le tableau 5.I résume le temps d'exécution moyen de l'heuristique DLAlign, calculé

sur 10 jeux de données, de 500 simulations (couples de séquences générées) chacun.

Paramètres (n, l=n/10)		Temps moyen
(500,50)		0.018
(1000,100)		0.088
(2000,200)		0.543
(3000,300)		0.566
(4000,400)		0.983
n=5000	l	
	n/20	1.585
	n/10	1.609
	n/8	1.596
	n/6	1.569
	n/4	1.651

Tableau 5.I : Temps d'exécution moyen, en secondes, pour chaque jeu de données de 500 simulations. Chaque jeu de données dépend des valeurs des paramètres n et l. La taille de l'alphabet est fixée à $a = n/10$ pour chaque jeu de données.

Bien que le temps d'exécution de DLAlign augmente légèrement avec la valeur de n, il ne dépasse pas deux secondes pour des séquences de taille $n = 5000$, et demeure stable dans le cas où le nombre l d'opérations augmente. Cependant, les données générées selon le premier choix des paramètres $n = 500$ et $l = 50$ peuvent prendre plusieurs minutes pour être traitées par l'algorithme DupLoCut. Les données générées selon le deuxième choix des paramètres $n = 1000$ et $l = 100$ peuvent prendre plusieurs heures, et les données générées selon le troisième choix des paramètres $n = 2000$ et $l = 200$ peuvent prendre plusieurs jours (plus de 11 jours) pour être traitées par l'algorithme DupLoCut

Notez que la faible augmentation du temps de calcul, due à l'augmentation du nombre l d'opérations lorsque la taille n des séquences est fixée à $n = 5000$, peut se justifier par le fait que, plus le nombre l augmente, plus le nombre de cycles dans l'alignement interprété est potentiellement grand, et nécessite donc un temps additionnel afin de corriger l'interprétation de l'alignement. Notez également que $n = 5000$ est la taille initiale du génome aléatoire, avant d'effectuer les opérations de duplications. Le génome final est

potentiellement de taille significativement supérieure à 5000.

5.3.4.2 Exactitude

Afin d'évaluer la qualité des solutions obtenues, nous avons comparé le coût *Res* de l'alignement interprété obtenu par DLAlign avec le coût optimal (minimum) *Opt* obtenu par l'algorithme exact DupLoCut. Deux mesures ont été considérées :

$Erreur = \frac{Res-Opt}{Res}$: représente le taux d'erreur sur le calcul du coût *Res* obtenu par l'heuristique DLAlign.

$Exact = \frac{NbOpt}{Total}$: représente le nombre $NbOpt$ des simulations, parmi le nombre total $Total$ des simulations, pour lesquelles le coût *Res* obtenu par l'heuristique DLAlign est minimum (*i.e.* $Erreur = 0$).

Dans cette expérience, on analyse l'impact de la taille de l'alphabet α et le nombre l d'opérations sur les résultats fournis par l'heuristique DLAlign.

Dans un premier temps, nous avons évalué la qualité des solutions obtenues par notre heuristique sur des séquences de petite taille ($n = 200$). Deux séries de jeux de données ont été utilisées. La première série porte sur la variation des rapports $Exact$ et $Erreur$ en fonction de la taille α de l'alphabet. La deuxième série porte sur la variation de ces deux rapports en fonction du nombre l d'opérations. De plus, le nombre de simulations de chaque jeu de données est fixé à $Total = 1000$.

Les deux paramètres l et α considérés dans les deux séries sont fixés respectivement à $2 * l/n = 1/5$ et $\alpha/n = 1/2$. Ces deux rapports sont similaires à ceux observés chez les Bacillus [149].

Les résultats de l'expérience sont donnés à la figure 5.6. Le diagramme gauche de la figure présente les résultats de la première série, et le diagramme droit présente les résultats de la deuxième série. Un bâton noir représente la valeur du rapport $Exact$, et un bâton blanc représente la valeur du rapport $Erreur$. Ces deux rapports sont moyennés sur le nombre total ($Total = 1000$) des simulations.

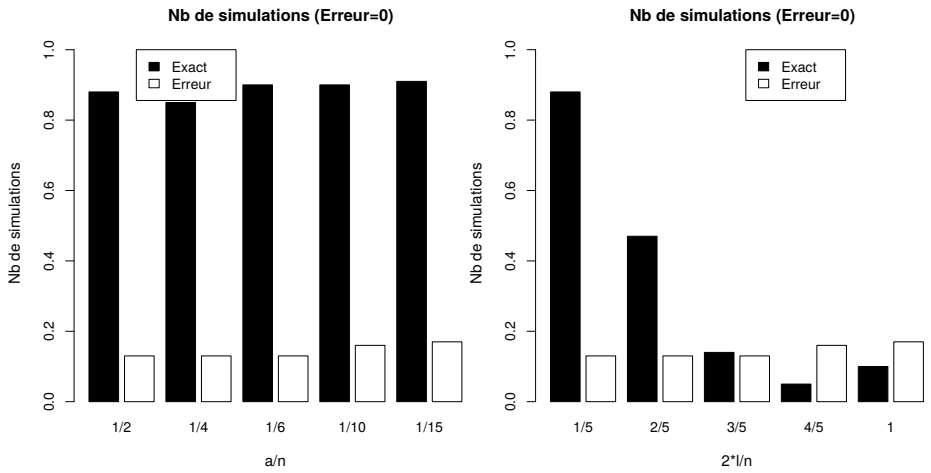


Figure 5.6 : Variation des rapports *Exact* et *Erreur* en fonction de la taille α de l'alphabet et du nombre l d'opérations sur 10 jeux de données de 1000 simulations chacun. Le diagramme gauche est obtenu en variant la taille α de l'alphabet (l'axe des x est α/n). Le diagramme droit est obtenu en variant le nombre l d'opérations (l'axe des x est $2*l/n$).

Avec les valeurs $2l/n = 1/5$ et $a/n = 1/2$, l'heuristique DLAlign retourne l'alignement interprété de coût minimum pour 85% des simulations (premier bâton de chaque diagramme). Ce taux reste stable en diminuant la taille de l'alphabet, *i.e.* en augmentant le nombre de copies de chaque gène (voir diagramme gauche), mais diminue rapidement avec l'augmentation du nombre l d'opérations (voir diagramme droit). En revanche, notez que même dans le cas où le nombre l d'opérations augmente jusqu'à la taille n des séquences (les séquences considérées sont donc très divergentes), le taux d'erreur moyen ne dépasse pas la valeur 0.16.

Par ailleurs, il est important de souligner que l'ancêtre construit à partir de l'alignement interprété réalisable de deux séquences X et Y n'est pas garanti aussi proche, en terme d'opérations de duplication et perte, de l'ancêtre réel des deux séquences considérées. En effet, pour deux génomes donnés, il existe autant d'ancêtres que d'alignements interprétés de coût minimum.

Nous avons ensuite réalisé la même expérience mais en considérant cette fois-ci des séquences de taille plus grande ($n = 1000$). De plus, le nombre de simulations de chaque

jeu de données est fixé à $Total = 50$. Les résultats de cette expérience sont donnés à la figure 5.7.

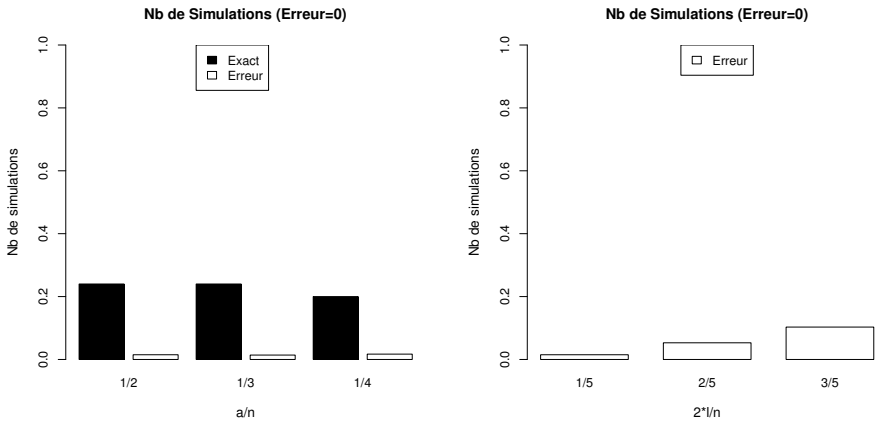


Figure 5.7 : Variation des rapports *Exact* et *Erreur* en fonction de la taille α de l'alphabet et du nombre l d'opérations sur 3 jeux de données de 50 simulations chacun.

Augmenter la taille des séquences à $n = 1000$ a pour conséquence la diminution du taux d'exactitude. En effet, avec les valeurs $2l/n = 1/5$ et $a/n = 1/2$, l'heuristique DLAlign retourne l'alignement interprété de coût minimum pour 24% des simulations. Ce taux reste stable en diminuant la taille de l'alphabet (voir diagramme gauche), mais diminue très rapidement et devient nul avec l'augmentation du nombre l d'opérations (voir diagramme droit). En revanche, le taux d'erreur moyen ne dépasse pas la valeur 0.103.

5.3.5 Conclusion

Dans ce chapitre, nous avons étudié le problème IMA d'interprétation minimale d'un alignement de deux génomes par duplications et pertes. Nous avons montré que ce problème est **APX-Difficile** même dans le cas où le nombre de copies de chaque gène dans le génome ne dépasse pas 5. En revanche, le problème admet une solution polynomiale dans le cas où le nombre de copies de chaque gène est au plus 2.

Ensuite, nous avons présenté deux heuristiques. La première porte sur la résolution

du problème RMA de réinterprétation minimale d'un alignement de deux génomes. La seconde heuristique DLAlign traite du problème d'alignement de deux génomes par duplications et pertes. DLAlign est basée sur la programmation dynamique et est quadratique en temps et en espace.

Les résultats expérimentaux présentés montrent nettement la qualité de DLAlign en terme de solutions fournies, et son efficacité en terme de temps d'exécution par rapport à l'algorithme exact PBLP.

Cette efficacité permet une application plus générale à un ensemble de génomes. En effet, l'algorithme d'alignement de deux ordres de gènes peut être utilisé dans un contexte plus large pour l'inférence d'ordres ancestraux d'un ensemble d'ordres de gènes dans un arbre phylogénétique donné. Ces perspectives sont abordées dans les prochains chapitres.

CHAPITRE 6

ORTHOALIGN, HEURISTIQUE POUR LE PROBLÈME D'ALIGNEMENT DE DEUX GÉNOMES PAR DUPLICATIONS, PERTES, RÉARRANGEMENTS ET SUBSTITUTIONS

6.1 Introduction

Dans le chapitre précédent, nous avons présenté DLAlign, une heuristique pour le problème d'alignement de deux génomes par duplications et pertes. Nous avons montré son efficacité en terme de résultats fournis et en temps d'exécution par rapport à l'algorithme exact PBLP de complexité exponentielle. Dans ce chapitre, nous proposons OrthoAlign, une généralisation de DLAlign aux opérations de réarrangement (transposition, transposition inversée, inversion), duplication inversée et de substitution. Même si l'on perd le caractère quadratique de l'heuristique, la complexité de cette dernière est cubique en la taille des deux génomes en entrée.

Récemment, nous avons étudié l'évolution de répertoire des gènes d'ARN de transfert chez 50 souches de *Bacillus*, du point de vue de leur nombre et de leur organisation dans le génome, et ceci dans le but de comprendre les mécanismes évolutifs qui sont les plus fréquents et l'origine de nouveaux gènes d'ARN de transfert. L'heuristique OrthoAlign a été utilisée dans le cadre de cette analyse.

Ce chapitre est organisé comme suit. Nous présentons, dans la section 6.2, l'heuristique OrthoAlign. Ensuite, dans la section 6.3, nous présentons une stratégie de reconstruction de séquences ancestrales dans un arbre phylogénétique donné, en se basant sur l'heuristique OrthoAlign. Enfin, la dernière section 6.4 porte sur les applications et validation de OrthoAlign sur des jeux de données simulés.

6.2 Description de l'heuristique

Comme OrthoAlign est une généralisation de l'heuristique DLAlign aux opérations de réarrangement (inversion, transposition, transposition inversée, duplication inversée) et de substitution, nous détaillons ici uniquement la description des équations de récurrence permettant le calcul de l'alignement interprété de coût minimum.

6.2.1 Extension de la programmation dynamique dans OrthoAlign

Soient X et Y deux séquences, et C une matrice de taille $|X|.|Y|$ telle que chaque entrée $C[i,j]$ représente le coût minimum de l'alignement interprété des deux préfixes $X[1,i]$ et $Y[1,j]$. On considère 9 situations pour la dernière colonne de l'alignement : un match, une perte dans le génome X, une perte dans le génome Y, une duplication (ou duplication inversée) dans le génome X, une duplication (ou une duplication inversée) dans le génome Y, une substitution de $X[i]$ par $Y[j]$ (ou vice-versa), une inversion d'un segment incluant $X[i]$ (ou un segment incluant $Y[j]$). Ces coûts sont respectivement représentés par $M(i,j)$, $P_X(i,j)$, $P_Y(i,j)$, $D_X(i,j)$, $DI_X(i,j)$, $D_Y(i,j)$, $DI_Y(i,j)$, $S[i,j]$, $I[i,j]$, calculés par les équations de récurrence suivantes.

- $M(i,j) = \begin{cases} C(i-1,j-1) & \text{si } X[i] = Y[j] \\ +\infty & \text{sinon} \end{cases}$

- $P_X(i,j) = min_{0 \leq k \leq j-1}[C(i,k) + c(L(j-k))]$

- $P_Y(i,j) = min_{0 \leq k \leq i-1}[C(k,j) + c(L(i-k))]$

- $D_X(i,j) = \begin{cases} +\infty & \text{si } X[i] \text{ est un "singleton"} \\ min_{l \leq k \leq i-1}[C(k,j) + c(D(i-k))] & \text{sinon} \end{cases}$
  où $X[l,i]$ est le plus long suffixe de $X[1,i]$ ayant une copie dans X ou dans Y.

- $DI_X(i,j) = \begin{cases} +\infty & \text{si } X[i] \text{ est un “singleton”} \\ min_{l \leq k \leq i-1}[C(k,j) + c(ID(i-k))] & \text{sinon} \end{cases}$

  où $X[l,i]$ est le plus long suffixe de $X[1,i]$ tel que $-X[l,i]$ est présent dans X ou dans Y.

- $D_Y(i,j) = \begin{cases} +\infty & \text{si } Y[j] \text{ est un “singleton”} \\ min_{l \leq k \leq j-1}[C(i,k) + c(D(j-k))] & \text{sinon} \end{cases}$

  où $Y[l,j]$ est le plus long suffixe de $Y[1,j]$ ayant une copie dans X ou dans Y.

- $DI_Y(i,j) = \begin{cases} +\infty & \text{si } Y[j] \text{ est un “singleton”} \\ min_{l \leq k \leq j-1}[C(i,k) + c(ID(j-k))] & \text{sinon} \end{cases}$

  où $Y[l,j]$ est le plus long suffixe de $Y[1,j]$ tel que $-Y[l,j]$ est présent dans X ou dans Y.

- $S(i,j) = \begin{cases} C(i-1,j-1) + c(S(1)) & \text{si } X[i] \neq Y[j] \\ +\infty & \text{sinon} \end{cases}$

- $I(i,j) = \begin{cases} min_{k \in E}[C(k-1, j-(i-k)-1) + c(I(i-(k-1)))] & \text{si } E \neq \emptyset \\ +\infty & \text{sinon} \end{cases}$

  où E est l'ensemble $\{k_1, k_2, \cdots k_{l \leq i}\}$ de cardinalité maximale tel que $X[k_p, i]$ est l'inverse de $Y[j-(i-k_p), j]$ pour tout $k_p \in E$.

Le coût minimum $C[i,j]$ de l'alignement interprété des deux préfixes $X[1,i]$ et $Y[1,j]$ est alors le minimum entre les différentes valeurs de ces équations. Les cas de base sont donnés par les équations suivantes :

- $C(0,0) = 0$
- $M(i,0) = M(0,j) = +\infty$
- $P_X(i,0) = P_Y(0,j) = +\infty$

- $D_X(0,j) = DI_X(0,j) = D_Y(i,0) = DI_Y(i,0) = +\infty$
- $S[i,0] = S[0,j] = I[i,0] = I[0,j] = +\infty$

Les équations de récurrence précédentes permettent de calculer le coût minimum de l'alignement interprété des deux génomes X et Y. Pour déterminer un des alignements interprétés possibles, il est nécessaire de fixer la source de chaque opération symétrique (transposition, transposition inversée, inversion ou substitution) dans l'un des deux génomes.

6.2.2 Correction de l'interprétation de l'alignement non-réalisable

Rappelons que l'alignement interprété obtenu par les équations de récurrence précédentes n'est pas garanti réalisable car il peut contenir un ou plusieurs cycles de duplications. La procédure effectuée pour limiter la possibilité d'avoir de tels cycles se déroule de la même manière que dans l'heuristique DLAlign, en fixant toujours, si c'est possible, la source des duplications dans le même génome.

La correction de l'interprétation non-réalisable se base sur l'heuristique pour le problème RMA discuté dans le chapitre précédent, à la différence près que le graphe de duplications à construire est associé à toutes les duplications de X et Y. En effet, les deux génomes alignés ne peuvent pas, cette fois, être traités séparément, en raison de la présence d'opérations de réarrangement et de substitution. De plus, tout cycle composé de deux duplications, ayant chacune une source et une cible n'appartenant pas au même génome, correspond à une transposition ou transposition inversée et sera donc supprimé entièrement du graphe. Les deux duplications correspondantes seront remplacées, dans l'interprétation résultante, par une seule transposition ou transposition inversée selon si ces duplications sont inversées ou pas (voir figure 6.1 pour une illustration).

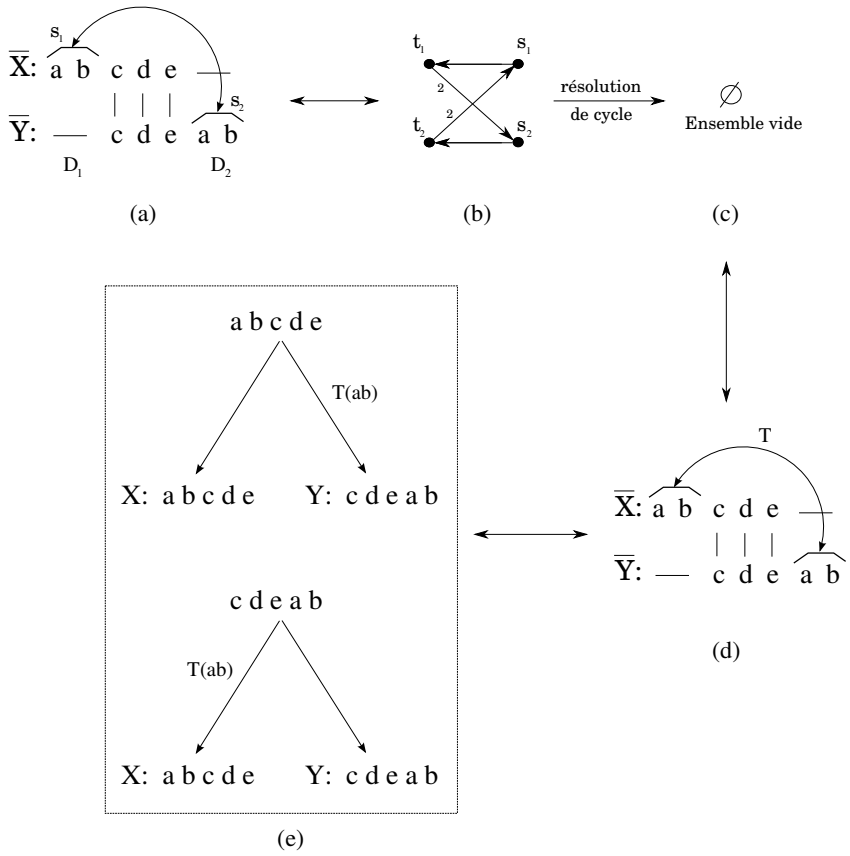


Figure 6.1 : Inférence d'une transposition dans un alignement. (a) Alignement interprété de deux génomes $X =$ "abcde" et $Y =$ "cdeab". (b) Cycle associé aux deux duplications D_1 et D_2 correspondant à une transposition. (c) Graphe résultant après suppression du cycle montré en (b). (d) Alignement interprété résultant après suppression du cycle montré en (b). (e) Deux histoires évolutives possibles selon le génome cible de la transposition $T(ab)$ des gènes "ab".

6.2.3 Analyse de complexité

Pour simplifier, posons $|X| = |Y| = n$. Clairement, les valeurs des équations de récurrence précédentes $M[i, j]$ et $S[i, j]$ se calculent en temps constant. Pour les équations $D_X[i, j]$, $DI_X[i, j]$, $D_Y[i, j]$ et $DI_Y[i, j]$, tous les plus longs suffixes des préfixes de X et Y, ayant une copie ou un inverse dans X ou Y peuvent se calculer de la même façon que dans l'heuristique DLAlign, et donc peuvent être retrouvés en temps constant. Par conséquent, les valeurs de ces équations se calculent en temps linéaire. Enfin, pour l'équation $I[i, j]$, toutes les inversions possibles de tous les préfixes de X et Y peuvent se calculer en

temps cubique dans une phase de pré-traitement. Le calcul des valeurs de cette équation se fait donc en temps linéaire. Par conséquent, la complexité pour calculer la matrice entière est de l'ordre de $O(n^3)$. La correction de l'interprétation non-réalisable se fait en temps $O(n^2)$ (voir section 6.2.2). La complexité de l'heuristique est donc de l'ordre de $O(n^3)$.

6.3 Application de OrthoAlign au problème de la petite phylogénie

Dans cette section, nous proposons une stratégie, basée sur l'utilisation directe de l'heuristique OrthoAlign, pour la reconstruction des séquences ancestrales aux noeuds internes d'un arbre phylogénétique donné.

L'idée est de résoudre le problème de la petite phylogénie d'une cerise au niveau de chaque noeud interne de l'arbre. Les noeuds internes de l'arbre sont visités niveau par niveau suivant une approche ascendante (*i.e.* depuis les feuilles jusqu'à la racine), et pour chaque noeud A visité, on applique OrthoAlign sur ses deux descendants X et Y.

Toutefois, il est à noter que l'heuristique OrthoAlign donne lieu à une multitude d'histoires évolutives parcimonieuses (*i.e.* de coût minimum), conduisant à plusieurs ancêtres possibles. Ceci est dû en particulier au fait que les opérations symétriques (inversion, transposition et substitution) sont applicables indifféremment à l'un ou l'autre des deux génomes. De plus, dans certains cas, les gènes peuvent être associés indifféremment, dans un alignement donné, à une duplication ou une perte avec le même coût de l'alignement. Afin de pouvoir identifier correctement le génome qui a subit l'opération symétrique en question, et de distinguer entre le type d'opération (duplication ou perte) associée aux gènes des deux génomes alignés, nous considérons, dans un post-traitement, le "génome voisin" d'une cerise.

Ce post-traitement se déroule comme suit : à partir de l'alignement interprété $\mathscr{L}(\overline{X},\overline{Y})$ des deux génomes X et Y, on détermine d'abord l'alignement interprété du génome X (ou Y, supposons que ce soit X sans perte de généralité) avec son voisin W le plus proche

dans l'arbre. Ensuite, pour tout segment $X[i,i+k]$ ($k \geq 0$) de X associé à une opération symétrique dans $\mathscr{L}(\overline{X},\overline{Y})$, si tous les gènes de $X[i,i+k]$ appartiennent à des colonnes "match" dans l'alignement $\mathscr{L}(\overline{X},\overline{W})$, alors $X[i,i+k]$ sera considéré comme la source de l'opération en question. Dans le cas contraire (*i.e.* un ou plusieurs gènes de $X[i,i+k]$ sont associés à des opérations), $X[i,i+k]$ est considéré comme la cible. De même, pour toute source $D^S[i,i+k]$ ($k \geq 0$) de duplication dans X, si tous les gènes de $D^S[i,i+k]$ appartiennent à des colonnes "match" dans l'alignement $\mathscr{L}(\overline{X},\overline{W})$ alors, $S[i,i+k]$ ($k \geq 0$) sera associé à une perte plutôt qu'à une duplication (voir la figure 6.2 pour une illustration).

6.4 Application et Validation

6.4.1 Analyse de l'évolution des gènes d'ARN de transfert chez 50 souches de *Bacillus*

Les ARN de transfert (ou ARNt) constituent une classe importante d'ARN nécessaire à la synthèse des protéines. Ils sont indispensables à la traduction de l'information codée par les séquences des ARN messagers (ARNm).

L'heuristique OrthoAlign a été utilisée pour l'analyse de l'évolution des gènes d'ARNt chez 50 souches de bactéries du genre *Bacillus*.

Plusieurs aspects ont été abordés lors de cette analyse, dont l'ordre et le contenu en gènes d'ARNt des séquences ancestrales, le type et le taux moyen des événements inférés, la distribution des tailles des différents événements et le taux des événements inférés. Notez que l'inférence des séquences ancestrales est basée sur la stratégie présentée dans la section 6.3. Toutefois, certaines corrections supplémentaires ont été effectuées manuellement afin d'obtenir les séquences ancestrales les plus pertinentes.

La figure 6.3 donne une représentation condensée de la phylogénie étudiée, dans le sens où seulement les branches de l'arbre, dans lesquelles les événements majeurs ont été observés, sont montrées.

Cette analyse a conduit à des résultats intéressants. En effet, un total de 141 pertes,

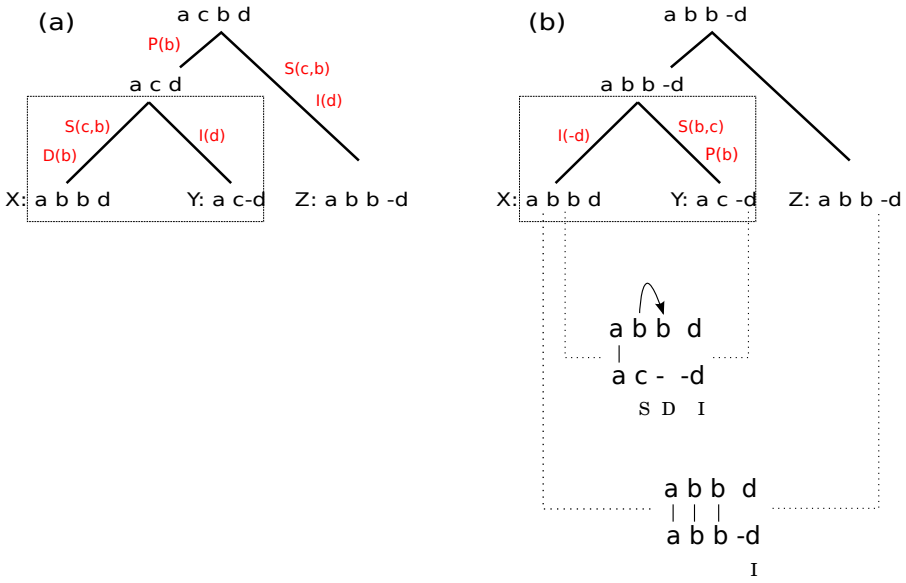


Figure 6.2 : Deux histoires évolutives possibles pour la phylogénie des trois génomes X, Y et Z. Dans le cas de comparaison des deux génomes de la cerise, délimitée par la zone en pointillés, deux histoires évolutives de trois opérations chacune peuvent être inférées, selon le génome cible des opérations. Dans l'histoire évolutive montrée en (a), la source de la substitution et celle de l'inversion se trouvent respectivement dans Y et X. De plus, il y a eu une duplication du gène "b". Tandis que dans l'histoire évolutive montrée en (b), la source de la substitution et celle de l'inversion se trouvent respectivement dans X et Y. De plus, une perte du gène "b" a été inférée au lieu d'une duplication. Ce deuxième scénario est obtenu en se servant du troisième génome Z voisin dans la phylogénie. Toutefois, dans le cas où toute la phylogénie est considérée, le scénario montrée en (b) est, clairement, le plus parcimonieux puisque trois événements sont suffisants pour expliquer toute l'histoire évolutive.

95 duplications, 23 inversions, 12 transpositions et 2 substitutions ont été inférées dans toute la phylogénie, montrant une évolution de ces gènes principalement par duplications et pertes. De plus, on a estimé que le taux des événements évolutifs dans le cas des gènes d'ARN de transfert chez *Bacillus* est beaucoup plus faible que celui des ARN de transfert d'organismes similaires comme la bactérie *Escherichia coli*. En effet, le taux des duplications (resp. des pertes) obtenu chez les souches de *Bacillus* est 24 fois (resp. 12 fois) plus faible que celui obtenu chez les souches de *Escherichia coli*.

Notez que l'interprétation détaillée des résultats obtenus lors de cette analyse sont au delà des objectifs de cette thèse. Les résultats rapportés ici serviront uniquement à évaluer à la fois la pertinence des résultats et les performances de OrthoAlign. Pour plus de détails, nous référons le lecteur à [243].

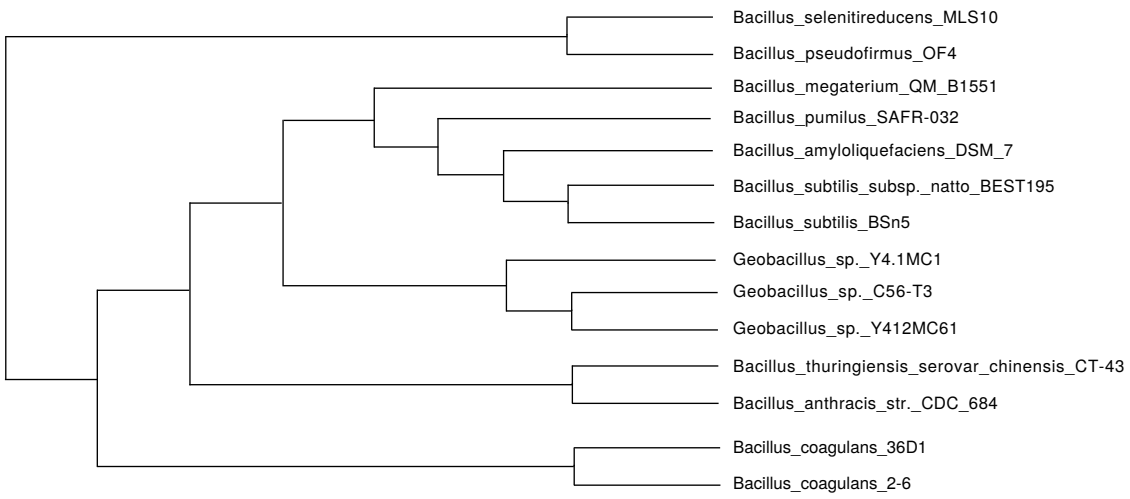


Figure 6.3 : Représentation condensée de la phylogénie étudiée, se limitant seulement aux branches de l'arbre dans lesquelles les événements majeurs ont été observés.

6.4.2 Validation des résultats sur des données simulées

Afin de valider les résultats de l'analyse discutée dans la section précédente, et de mesurer les performances de notre heuristique OrthoAlign, nous avons testé cette dernière sur des jeux de données simulés cohérents avec les répertoires des ARN de transfert étudiés.

Chaque jeu de données comporte plusieurs simulations d'arbres phylogénétiques, de même topologie, générés comme suit : La racine R de l'arbre, d'une centaines de gènes (nombre typique d'ARN de transfert chez les génomes de *Bacillus*) et d'un alphabet de taille 21 (nombre de familles différentes où une famille représente tous les ARNt associés à un acide aminé donné), est d'abord générée. Ensuite, toutes les séquences descendantes sont obtenues en effectuant l opérations (non-chevauchantes) sur leurs ancêtres directs. En s'inspirant des résultats observés pour les génomes de *Bacillus*, les opérations sont générées en suivant une loi géométrique de paramètre $p = 0.5$ pour la taille des opérations. Quant aux nombres n_1, n_2, n_3, n_4, n_5 d'opérations respectivement de perte, duplication (ou duplication inversée), inversion, transposition (ou transposition inversée) et substitution, ils sont générés suivant une loi multinomiale de paramètres

respectivement $p_1 = 0.516, p_2 = 0.347,\ p_3 = 0.084,\ p_4 = 0.043$ et $p_5 = 0.01$.

Nous nous focalisons ici sur l'évaluation des performances de OrthoAlign sur l'inférence du nombre correct d'opérations, les séquences ancestrales et la distribution des tailles des opérations dans de petites et de grandes phylogénie. Nous analysons, pour cela, l'impact du nombre l d'opérations, simulées dans chaque branche de l'arbre (*i.e.* entre un ancêtre et son descendant direct), sur les résultats fournis par notre heuristique dans plusieurs jeux de données. Chaque jeu de données possède une configuration distincte selon la valeur du paramètre l.

Deux mesures ont été utilisées. La première, définie par $Erreur = \frac{NbR-NbI}{NbR}$, représente le taux d'erreur sur l'inférence du nombre correct d'opérations dans l'histoire évolutive obtenue par OrthoAlign, où NbR est le nombre total d'opérations réelles (*i.e.* simulées) dans l'arbre, et NbI est le nombre total d'opérations inférées par OrthoAlign. La seconde mesure représente la distance évolutive (nombre d'opérations) entre l'ancêtre inféré par OrthoAlign et l'ancêtre réel (*i.e.* simulé) dans chaque noeud de l'arbre. Cette distance est obtenue par le calcul de l'alignement interprété des deux ancêtres considérés.

6.4.2.1 Évaluation sur de petites phylogénies :

Cette Expérience porte sur la variation du rapport $Erreur$ en fonction du paramètre l. Une série de 10 jeux de données, de 500 simulations (arbres générés) chacun, a été utilisée à cet effet. Le taux d'erreur est moyenné sur le nombre total de simulations de chaque jeu de données.

Les arbres considérés sont des cerises accompagnées d'un seul frère (voir figure 6.2). Les génomes des trois feuilles de l'arbre sont générés suivant la procédure discutée précédemment. Pour chaque arbre généré, on détermine le nombre total d'événements inférés et les deux séquences ancestrales des deux noeuds internes. Le génome de la troisième feuille est utilisé par l'heuristique OrthoAlign pour inférer correctement les

séquences ancestrales, comme mentionner dans la section 6.3.

Les résultats de cette expérience sont consignés sur les graphiques de la figure 6.4. On constate que le taux d'erreur moyen augmente avec l'augmentation du nombre l d'opérations simulées par branche de l'arbre.

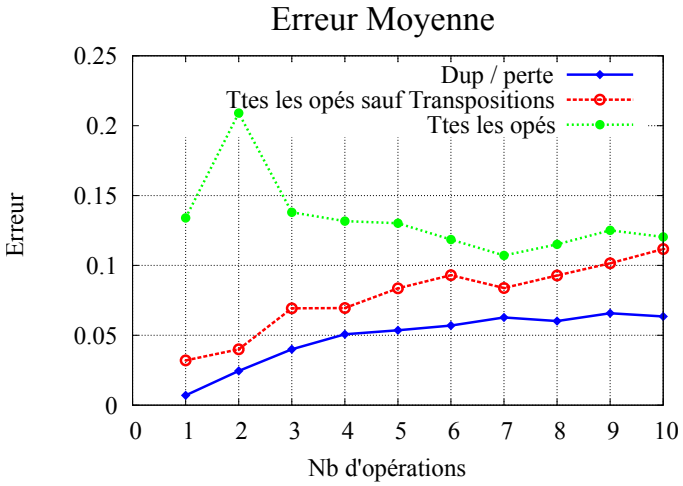


Figure 6.4 : Variation du taux d'erreur en fonction du nombre l d'opérations simulées dans chaque branche de l'arbre. Pour chaque valeur de l, correspond un jeu de données de 500 simulations. Les résultats sont moyennés sur 500 simulations. Le graphique bleu réfère au modèle évolutif restreint aux duplications et pertes, le rouge réfère au modèle évolutif impliquant toutes les opérations sauf les transpositions, et le vert réfère au modèle évolutif impliquant toutes les opérations y compris les transpositions.

Pour un modèle évolutif restreint aux opérations de duplication et perte (ligne bleue du graphique), ce taux d'erreur est très proche de 0, et les ancêtres inférés correspondent parfaitement aux ancêtres simulés pour $l \leq 3$, qui est le nombre moyen d'opérations de duplication et perte à chaque branche, observé chez les *Bacillus* [243].

Intégrer les inversions et les substitutions dans le modèle évolutif (ligne rouge du graphique) a pour conséquence une augmentation légère du taux d'erreur moyen, mais en revanche, on observe la même tendance que le modèle évolutif restreint aux duplications et pertes.

Cependant, le taux d'erreur moyen augmente significativement dans le cas du modèle évolutif qui tient compte de toutes les opérations y compris les transpositions (ligne verte du graphique). Dans ce cas, l'heuristique a forte tendance à inférer moins d'opéra-

tions dans l'arbre. Ceci est justifié par le fait que deux duplications dans deux génomes différents, ayant les mêmes sources et les mêmes cibles, peuvent par ailleurs être expliquées par une seule transposition. Notons que le pic observé, dans le cas où $l = 2$, est dû au fait que le numérateur varie en général entre 0 et 2 et le dénominateur est fixé à la valeur 8.

6.4.2.2 Évaluation sur de grandes phylogénies :

Cette seconde expérience porte sur la variation de la distance, calculée entre l'ancêtre inféré et l'ancêtre réel dans chaque noeud de l'arbre en fonction du paramètre l. Les arbres considérés, cette fois, possèdent la topologie montrée en figure 6.3. La distance est moyennée à la fois sur le nombre total des noeuds internes de l'arbre et sur le nombre total des simulations de chaque jeu de données.

La figure 6.5 donne la distance moyenne calculée sur 10 jeux de données de 50 simulations chacun.

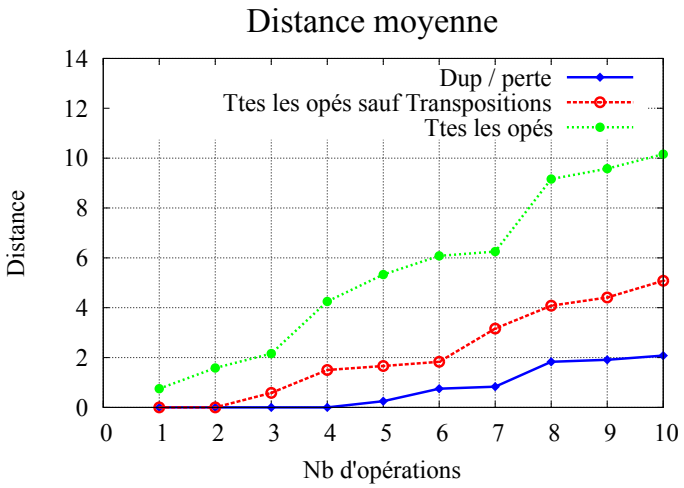


Figure 6.5 : Variation de la distance, calculée entre l'ancêtre inféré et l'ancêtre réel dans chaque noeud de l'arbre, en fonction du nombre l d'opérations simulées dans chaque branche de l'arbre. Pour chaque valeur de l, correspond un jeu de données de 50 simulations. Les résultats sont moyennés à la fois sur le nombre total de noeuds internes de l'arbre et sur le nombre total de simulations de chaque jeu de données.

À partir de cette figure, on constate que les meilleurs résultats sont obtenus avec le modèle évolutif restreint aux duplications et pertes (ligne bleue du graphique). En effet,

pour $l \leq 4$, l'algorithme infère correctement (distance nulle entre les deux ancêtres) les séquences ancestrales dans tout l'arbre. De plus, pour $l \leq 10$, la distance moyenne entre l'ancêtre réel et l'ancêtre inféré ne dépasse pas la valeur 2. Cependant, en intégrant les inversions, les transpositions et les substitutions dans le modèle évolutif (ligne rouge et ligne verte du graphique), les ancêtres inférés sont significativement différents de ceux simulés. Ceci est justifié par la multitude de solutions possibles dues aux différentes opérations considérées. En revanche, la distance moyenne entre l'ancêtre inféré et l'ancêtre réel ne dépasse pas 3 pour $l \leq 3$ (nombre moyen d'opérations sur chaque branche de l'arbre, observé chez les *Bacillus*).

6.4.2.3 Distribution de la taille des duplications et pertes :

Dans cette dernière expérience, nous évaluons les performances de OrthoAlign sur l'inférence correcte de la distribution des tailles des opérations de duplication et perte, qui sont les opérations les plus fréquentes observées dans la phylogénie étudiée.

La figure 6.6 donne la distribution des tailles des duplications et pertes inférées et simulées dans tout l'arbre pour $l = 7$.

Dans l'ensemble, la taille des opérations inférées est similaire à celle des opérations simulées. Toutefois, on remarque une surestimation du nombre de duplications de taille 1 et 2, et une légère sous-estimation du nombre de pertes de taille 1. Ceci montre que OrthoAlign a tendance à confondre parfois des pertes avec des duplications. Notez que le fait d'inférer une seule duplication, par erreur, au lieu d'une perte peut donner lieu à beaucoup plus de duplications dans l'histoire évolutive inférée dans toute la phylogénie dans le cas où l'erreur se produit dans les niveaux les plus profonds de l'arbre (comme par exemple le niveau des feuilles).

En résumé, ces expériences montrent que OrthoAlign infère, avec une bonne précision, les séquences ancestrales, le nombre et la distribution des tailles des opérations sur des jeux de données simulés cohérents avec le répertoire des ARN de transfert étudiés.

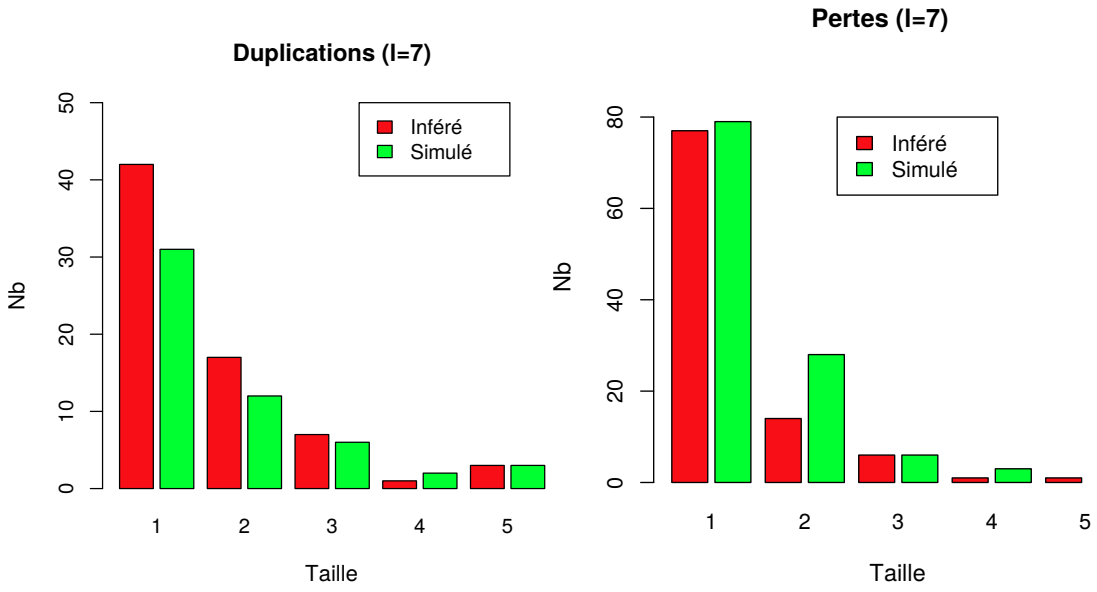


Figure 6.6 : Distribution des tailles des duplications et pertes, inférées et simulées dans l'arbre pour $l = 7$.

6.5 Conclusion

Dans ce chapitre, nous avons présenté OrthoAlign, une nouvelle heuristique de programmation dynamique, en temps cubique, pour le problème d'alignement de deux génomes représentés par des ordres de gènes. Le modèle évolutif considéré tient compte, en plus des duplications et pertes, des duplications inversées, des réarrangements (transpositions, transpositions inversées, inversions) et des substitutions.

OrthoAlign a été utilisée pour l'analyse de l'évolution des gènes d'ARNt chez 50 souches de bactéries du genre *Bacillus*. Les résultats de cette analyse montrent que ces gènes ont principalement évolué par duplications et pertes.

Les résultats de nos simulations montrent une bonne adéquation entre les données simulées et les résultats obtenus par OrthoAlign, en terme d'inférence de séquences ancestrales, de calcul du nombre et de la distribution des tailles des opérations, sur des jeux de données simulés cohérents avec les répertoires des ARNt des *Bacillus*. Ceci dit, la correction par le voisin direct d'une cerise est une approche phylogénétique assez ru-

dimentaire, qu'il serait intéressant d'améliorer et de formaliser. C'est le sujet du dernier chapitre de cette thèse.

CHAPITRE 7

GENE ORDER ALIGNMENT ON TREES WITH MULTIORTHOALIGN

Billel Benzaid[1] **and Nadia El-Mabrouk**[1]

Article publié dans la conférence RECOMB-CG 2014 [32].

7.1 Contribution

Billel Benzaid a conçu l'heuristique multiOrthoAlign pour la reconstruction de séquences ancestrales dans un arbre phylogénétiques donné. **Billel Benzaid** a implémentée l'heuristique, conçu et réalisé les expériences sur les données biologiques et les données simulées. **Billel Benzaid** et Nadia El-Mabrouk ont rédigé l'article.

[1]Département d'Informatique et de Recherche Opérationnelle DIRO, Université de Montréal, Canada

7.2 Abstract

We relate the comparison of gene orders to an alignment problem. Our evolutionary model accounts for both rearrangement and content-modifying events. We present a heuristic based on dynamic programming for the inference of the median of three genomes and apply it in a phylogenetic framework. multiOrthoAlign is shown accurate on simulated and real datasets, and shown to significantly improve the running-time of DupLoCut, an "almost" exact algorithm based on linear programming, developed recently for the same problem.

7.3 Introduction

A major requirement in comparative genomics is to be able to compare genomes based on their whole content. This is necessary for a myriad of applications such as phylogenetic reconstruction, orthology and paralogy identification, ancestral reconstruction and the study of evolutionary events. Consequently, a large variety of algorithms have been developed for the comparison of whole-genome sequences with partial or no information on gene annotation. Most of them are based on first identifying, in a pair-wise alignment dotplot, local alignments (anchors, syntenies) with high similarity score, and then chaining them in a way maximizing an alignment score (cf. e.g. MUMmer [89], BLASTZ [223], LAGAN [58], DAGchainer [134], progressiveMauve [87]). Similarity scores are computed according to the local mutations (nucleotide substitutions and indels) inferred from the alignment. Other approaches compare genomes in terms of building block organization. Although a recently developed method does not require any preliminary information on gene families [49], most of them assume a full or partial annotation of genomes, or a previously established large coverage of genomes in terms of syntenic blocks. Given two genomes represented as ordered sequences of genes (or building blocks), the rearrangement approach consists in finding a sequence of global evolutionary events transforming one gene order to the other. Early work on

genome rearrangement focused on sorting permutations (no duplicates) by rearrangements (inversions, translocations, transpositions) [137, 138, 156]. More recently, a variety of studies have considered the more difficult case of genomes with duplicates evolving through rearrangements, but also through content modifying operations such as duplications and losses (reviews in [109, 118]). Other model-free approaches based on conserved synteny, with no assumption on the evolutionary mechanisms, have also been developed [33, 37, 49, 101, 142, 255].

In a recent set of papers [30, 99, 148] we related the comparison of two gene orders to an alignment problem : find an alignment between the two gene orders that can be interpreted by a minimum number of evolutionary events (rearrangements and content-modifying operations). Although alignments are *a priori* simpler to handle than rearrangements, this problem has been shown NP-hard for the duplication-loss model of evolution [13, 30, 99]. Exact exponential-time algorithms based on linear programming [13, 148] and a polynomial-time heuristic based on dynamic programming [30] have been developed for this model. Recently [243], we developed OrthoAlign (alignment of orthologs), a time-efficient heuristic for the gene order alignment problem, that extends the dynamic programming approach to a model including rearrangements (inversions and transpositions).

Sequence and gene order alignments are useful for ancestral inference purposes. As explained in [148], a "labeled" pair-wise gene order alignment can be translated into a common ancestor and an evolutionary scenario leading to the two compared gene orders. Such an alignment approach for ancestral inference is relevant if the two gene orders reflect enough conservation so that we can assume that only few events have occurred since the divergence of the lowest common ancestor of the two genomes. For such closely related species, events can be assumed to be non-overlapping (each gene involved in at most one event) and thus still visible in the alignment. The gene-order alignment approach has been shown useful to decipher the evolutionary mechanisms that have shaped

the tRNA gene repertoires of the bacterium *Bacillus* [148].

Here, we undertake the next step, which is using the alignment approach on a phylogeny : infer ancestral genomes identified with each speciation node of a phylogenetic tree. The alignment on a tree problem introduced by Sankoff *et al.* in [220], consists in finding assignments of internal nodes in a way minimizing the total branch length of the tree according to a given distance. The result is, not only a set of ancestral genomes, but also a multiple alignment for extant sequences. As trying all possibilities for internal node assignments is intractable, iterative heuristics on subtrees are usually considered, the most popular being the median-based heuristic [109, 168] : (1) find an initial assignment for internal nodes ; (2) in a post-order traverse of the tree, improve the assignment of each internal node u by considering the median of the leaf-assignments of the 3-star tree centered on u, *i.e.*, the tree formed by the three neighbouring nodes of u ; (3) repeat until no improvement on the tree distance can be made. In the case of genomes represented as gene orders, applying the exact 2-SPP (2-Small Phylogeny Problem) algorithm [148] or OrthoAlign [243] to the cherries of the phylogeny can be used for an initial assignment. As for the iterative step, an efficient algorithm for the median problem has to be found. Although NP-hard for most versions of the problem [110, 206, 240], efficient heuristics have been developed for various nucleotide and rearrangement distances. As for the duplication-loss model of evolution, DupLoCut, an "almost" exact algorithm based on linear programming has been presented in [13].

In this paper, we present multiOrthoAlign for the alignment of a set of gene orders related through a phylogenetic tree. It is based on a dynamic programming approach generalizing OrthoAlign [30, 243] to a 3-star tree, under a model involving a wide range of evolutionary events. multiOrthoAlign is compared with DupLoCut [13], the most closely related algorithm. Experiments on simulated and real datasets reveal similar accuracy for both algorithms, but with a significant improvement in running time for multiOrthoAlign.

7.4 Method

We consider uni-chromosomal genomes represented as strings of signed characters from an alphabet Σ, where each character represents a gene family. Each character may appear many times in a genome G, all such positions corresponding to genes belonging to the given gene family. The sign of a gene represents its transcriptional orientation. Let $X = x_1 x_2 \cdots x_n$ be a string. We call the *reverse* of X the string $-X = -x_n \cdots - x_2 - x_1$. We denote by $X[i, i+k]$ the *substring* of X formed by the consecutive genes of the interval $[i, i+k]$.

A *phylogeny* or *species tree* $\mathscr{S}$ for a set Γ of genomes is a tree with a one-to-one correspondence between the leaves of $\mathscr{S}$ and the species of Γ, reflecting the evolution of the genomes through speciation. Although the method developed in this paper does not require any assumption on the species tree, for ease of presentation, we consider binary and rooted phylogenies. An internal node of $\mathscr{S}$ corresponds to a speciation event and an assignment for that node corresponds to the genome at the moment of speciation. A *phylogenetic alignment* $\overline{\mathscr{S}}$ for $\mathscr{S}$ is the tree $\mathscr{S}$ augmented with an assignment of one string for each internal node of S. When no ambiguity, we will make no difference between a node and its assignment. Two nodes are *related* if they belong to the same path from a leaf of $\mathscr{S}$ to the root, and *unrelated* otherwise. For two related nodes $A \neq X$, A is an *ancestor* of X if A is closer to the root of $\mathscr{S}$ than X. For two unrelated nodes $X \neq Y$, they are *siblings* if they share the same parental node. A pair of siblings is called a *cherry*. Moreover, we call a *3-star* of $\mathscr{S}$ and we denote by $A|XY$ a star-tree with three leaves A, X, Y such that X and Y are two siblings in $\mathscr{S}$ and A is the immediate ancestor of the parent M of X and Y. M is called the center of $A|XY$.

7.4.1 The evolutionary model

We assume that present-day genomes have evolved from an ancestral genome through rearrangement and content-modifying events, each event (operation) acting on a uni-

chromosomal genome X and leading to a new uni-chromosomal genome Y. An operation is denoted by $O(k) = (O^S, O^T)$, where O is the operation type, k is the length of the operation, O^S is the *source*, *i.e.*, the substring affected by the event and O^T is the *target*, *i.e.*, the new substring resulting from the event. Characters of O^S and O^T are said to be *covered* by the operation. The mostly considered content-modifying operations are duplications and losses, where :

- A **Duplication** $D(k) = (D^S = X[i, i+k-1], D^T = Y[j, j+k-1])$, where $Y[j, j+k-1] = X[i, i+k-1]$, is an operation that copies the substring $X[i, i+k-1]$ of size k to a location j outside the interval $[i, i+k-1]$ (*i.e.* preceding i or following $i+k-1$) ;
- A **Loss** $L(k) = (X[i, i+k-1], \oslash)$ ($\oslash$ for empty string) is an operation that removes the substring $X[i, i+k-1]$ from genome X.

The mostly considered uni-chromosomal rearrangements are reversals and transpositions, where :

- A **Reversal** (or inversion) $R(k) = (X[i, i+k-1], Y[i, i+k-1])$, where $Y[i, i+k-1] = -X[i, i+k-1]$, is an operation that transforms the substring $X[i, i+k-1]$ into its reverse ;
- A **Transposition** $T(k) = (X[i, i+k-1], Y[j, j+k-1])$, where $Y[j, j+k-1] = X[i, i+k-1]$, is an operation that moves the substring $X[i, i+k-1]$ to another position j outside the interval $[i, i+k-1]$.

Denote by $\mathscr{O}$ the set of operation types. We will describe our approach for $\mathscr{O} = \{D, L, R, T\}$. Including other events, such as inverted duplications or inverted transpositions with target being the reverse of the source, insertions which are the counterparts of losses, or substitutions replacing a string with another of the same size, do not add any

complexity to the problem. Notice however that the more operations we include to the model, the more challenging is the problem of assigning appropriate operations costs.

Let $\mathscr{S}$ be a phylogeny and X, Y be two nodes of $\mathscr{S}$. If X and Y are related, say X is an ancestor of Y, then a *history* $O_{X\to Y}$ for X and Y is a sequence of events (possibly of length 0) transforming X into Y. Otherwise, if X and Y are unrelated, then a *history* for X and Y is a triplet $(A, O_{A\to X}, O_{A\to Y})$, where A is an assignment of the lowest-common ancestral node of X and Y. We call a *visible history for X and Y* a history where the source and target of each operation is a substring of X or Y.

Finally, let $A|XY$ be a 3-star of $\mathscr{S}$. A history for $A|XY$ is a quadruplet $(M, O_{A\to M}, O_{M\to X}, O_{M\to Y})$ where M is an assignment of the center of the 3-star. A *visible history for $A|XY$* is a history where the source and target of each operation is a substring of A, X or Y.

Notice that a duplication with source and target in two different genomes can be interpreted as a duplication followed by the loss of the source (a relaxation of visibility), or alternatively as a transposition, or even as a horizontal gene transfer between the two considered genomes. We will take this general view of a duplication, which implicitly integrates transpositions.

7.4.2 Genome alignment

We begin by recalling the classical notion of an alignment of strings (genomes) $\Gamma = \{X_k : 1 \leq k \leq \gamma\}$. Let $\Sigma^- = \Sigma \cup \{-\}$ be the alphabet Σ augmented with an additional character '-' called a gap. Then an *alignment* for Γ is a set $\overline{\Gamma} = \{\overline{X_k} : 1 \leq k \leq \gamma\}$ of strings obtained by filling X_k with gaps, such that the resulting *aligned genomes* have equal length λ, and for each position i, $1 \leq i \leq \lambda$, the column i is *not empty* in the sense that at least one of $\overline{X_k}[i]$, for $1 \leq k \leq \gamma$, is not a gap. The *induced alignment* for a subset $\Gamma' \subset \Gamma$ is the alignment $\overline{\Gamma'}$ obtained by removing from $\overline{\Gamma}$ all genomes that are not in Γ' and all empty columns. Given a pair $(\overline{X_l}[i], \overline{X_m}[i])$ of aligned characters, it is a *match* if $\overline{X_l}[i] = \overline{X_m}[i] \in \Sigma$, a *mismatch* if $\overline{X_l}[i] \neq \overline{X_m}[i]$ both being in Σ and a *gap* if $\overline{X_l}[i] \in \Sigma$ and

$\overline{X_m}[i]$ ='-'.

A multiple alignment is expected to reflect the evolutionary events that have led to the present-day genomes. The notion of an *alignment labeling* has been introduced in [148] for a pair-wise alignment. It relates each column of the alignment to a given operation. Generalization to an arbitrary number of genomes is given below. We will make use of this definition later in the context of a 3-star history.

Definition 1 *Let $\overline{\Gamma} = \{\overline{X_k} : 1 \leq k \leq \gamma\}$ be an alignment of length λ. A labeling $\mathscr{L}(\overline{\Gamma})$ for $\overline{\Gamma}$ is a set of operations covering the characters of the given sequences. For any l and m in $[1,\gamma]$ with $l \neq m$ and any i, $1 \leq i \leq \lambda$, such that $\overline{X}_l[i] \neq$ '$-$', $(\overline{X}_l[i], \overline{X}_m[i])$ is covered by at most one operation of $\overline{\Gamma}$ as follows :*

- *if a match, then it is covered by no operation ;*
- *if a mismatch, then it is covered by a reversal ;*
- *if a gap, then it is covered by one of the other operations of $\mathscr{O}$.*

with the restriction that, if the two genomes are related, say X_l is an ancestor of X_m, then the source of the operation should be in X_l and the target should be in X_m.

A *labeled alignment* is an alignment $\overline{\Gamma}$ accompanied with a labeling $\mathscr{L}(\overline{\Gamma})$. We simply refer to a labeled alignment by its labeling $\mathscr{L}(\overline{\Gamma})$. The *cost of a labeled alignment* is the sum of costs of all its labeling events.

The above definition does not ensure a valid interpretation of a labeled alignment in terms of an evolutionary history $(A, O_{A\to X}, O_{A\to Y})$ for two genomes X and Y. We showed in [148] that a pair-wise labeled alignment is valid if and only if it is free from cycles, where cycles are defined as follows.

Definition 2 *Let $\mathscr{O}$ be a set of operations. It induces a* cycle *if there is a permutation $O_1, O_2, \cdots O_h$ of $\mathscr{O}$ events such that the substrings O_p^T and O_{p+1}^S overlap (a suffix of O_p^T is a prefix of O_{p+1}^S), for each $1 \leq p \leq h-1$, and the substrings O_h^T and O_1^S overlap.*

A *feasible labeled alignment* is a labeled alignment with no cycles. We showed in [148] the one-to-one correspondence between feasible labeled alignments and visible histories for two genomes X and Y in case of an evolution through duplications and losses.

7.4.3 Phylogenetic alignment

Let $\mathscr{S}$ be a species tree for a genome set Γ. Call a *feasible labeled phylogenetic alignment* for $\mathscr{S}$ a phylogenetic alignment $\overline{\mathscr{S}}$ accompanied with a feasible labeled alignment for each cherry (X,Y) of $\overline{\mathscr{S}}$, in other words a visible history $(A, O_{A\to X}, O_{A\to Y})$ for each (X,Y). Such a feasible labeled phylogenetic alignment leads to a multiple alignment for Γ : traverse $\overline{\mathscr{S}}$ in post-order and iteratively incorporate alignments of cherries in a current multiple alignment which is initially empty.

Let A and X be two genomes of $\mathscr{S}$ with A being an ancestor of X and let $O_{A\to X} = \{O_1(k_1), \cdots O_m(k_m)\}$ be a history for A and X. The cost of $O_{A\to X}$ is defined as $C(O_{A\to X}) = \sum_{i=1}^{m} c(O_i(k_i))$, where $c(O_i(k_i))$ is the cost of the operation $O_i(k_i)$. Let $\mathscr{O}_{A\to X}$ be the set of all possible histories transforming A into X. We define $C(A\to X) = \min_{O_{A\to X}\in\mathscr{O}_{A\to X}} C(O_{A\to X})$. Now, the *phylogenetic alignment problem*, is to infer a feasible labeled phylogenetic alignment for $\mathscr{S}$ minimizing the sum of costs of all branches of $\mathscr{S}$.

The relaxed phylogenetic alignment problem with no restriction on visibility, *i.e.* the problem of assigning ancestral configurations leading to a minimum cost for the tree, has been shown to be NP-hard for most formulations in terms of type of genomes and different distances. A classical heuristic strategy is known as the *steinerization approach* [168]. It begins with an initial assignment for the internal nodes of $\mathscr{S}$, and in a post-order traversal it improves each internal node assignment by solving a 3-star problem defined as follows.

3-star Problem :

INPUT : A 3-star phylogeny $A|XY$.

OUTPUT : A visible history $(M, O_{A\to M}, O_{M\to X}, O_{M\to Y})$ for $A|XY$ minimizing the cost :

$$C(A \to M) + C(M \to X) + C(M \to Y).$$

In the case of symmetrical operations, such as nucleotide substitutions or indels, or gene order rearrangements, the direction of evolution can be ignored, which leads to the median problem : find M minimizing $C(M,A) + C(M,X) + C(M,Y)$. However, this is not the case for content-modifying operations, as for example a duplication from A to X is rather a loss from X to A, and therefore the evolutionary direction cannot be ignored in this case.

For the evolutionary model of interest, the restriction of the phylogenetic alignment problem to a cherry has been considered in [30, 148]. The developed algorithm can be used for the initialization step : traverse the tree in a depth-first manner and compute successive ancestors of pairs of nodes. Here, we extend our study to a 3-star phylogeny, which allows for the application of the aforementioned steinerization approach. Notice that the phylogenetic alignment problem has been shown NP-complete for the duplication-loss model of evolution, already for two species [13, 30, 99].

7.4.4 The 3-star Problem

We first show that the 3-star problem for a 3-star $A|XY$ reduces to finding a feasible labeled alignment for $\{A,X,Y\}$ of minimum cost. It is easy to see that any visible history for $A|XY$ leads to a unique feasible labeled alignment for $\{A,X,Y\}$. Conversely, let $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$ be a feasible labeled alignment for a 3-star $A|XY$. A corresponding visible history for $A|XY$ can be obtained as follows (see Figure 7.1 for an example) :

- Define $(M, O_{M\to X}, O_{M\to Y})$ as the visible history corresponding to the induced feasible labeled alignment for X and Y.

- Consider the alignment $(\overline{A},\overline{M})$, where $\overline{M}$ is the aligned genome M corresponding to the above history.
- Define $\mathscr{L}(\overline{A},\overline{M})$ as follows. For each i such that $(\overline{A}[i],\overline{M}[i])$ is not a match :
  - If $\overline{X}[i] = \overline{Y}[i]$ then include in $\mathscr{L}(\overline{A},\overline{M})$ the operation of $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$ covering the column $(\overline{A}[i],\overline{X}[i])$ (or alternatively $(\overline{A}[i],\overline{Y}[i])$).
  - Otherwise $\overline{M}[i]$ should be equal to $\overline{X}[i]$ or $\overline{Y}[i]$. Assume w.l.o.g. that $\overline{M}[i] = \overline{X}[i]$. Then include in $\mathscr{L}(\overline{A},\overline{M})$ the operation of $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$ covering the column $(\overline{A}[i],\overline{X}[i])$.

Therefore, given a 3-star $A|XY$, we focus here on the problem of finding a feasible labeled alignment for $\{A,X,Y\}$ of minimum cost.

Let $C(i,j,k)$ ($C^f(i,j,k)$ respectively) be the minimum cost of a labeled (feasible labeled respectively) alignment of three prefixes $A[1,i]$, $X[1,j]$ and $Y[1,k]$ of A, X and Y, for all $1 \leq i \leq |A|$, $1 \leq j \leq |X|$ and $1 \leq k \leq |Y|$. Step 1 described below gives a heuristic for computing $C(i,j,k)$ and Step 2 a heuristic for computing $C^f(|A|,|X|,|Y|)$ from $C(|A|,|X|,|Y|)$.

• STEP 1. FINDING A LABELED ALIGNMENT BY A DYNAMIC PROGRAMMING APPROACH.

As explained previously, transpositions are implicitly considered by allowing the source and target of a duplication to belong to two different genomes. Therefore, we will restrict our presentation to the model $\mathscr{O} = \{D,L,R\}$.

To compute $C(i,j,k)$, we consider all the possibilities for the last column of an alignment of the three prefixes $A[1,i]$, $X[1,j]$ and $Y[1,k]$ and interpret it by the minimum number of operations. In the following, a column is represented as a triplet of characters from Σ^-, were different letters denote different characters of Σ. Clearly, each column can be interpreted by no more than 2 operations. If two operations are required to interpret a

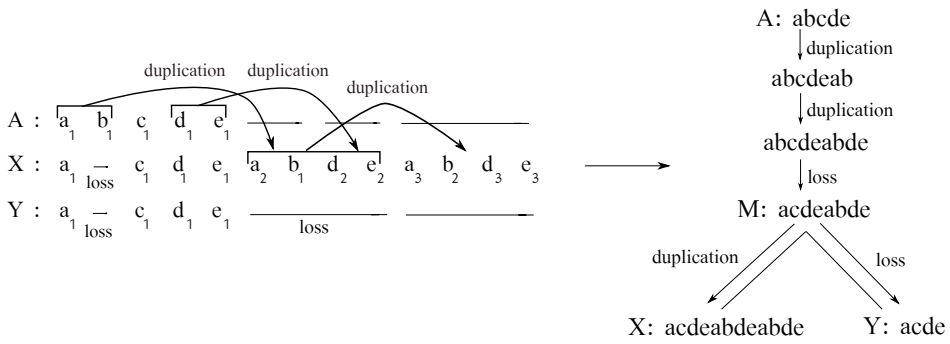


Figure 7.1 : Left : a labeled alignment for strings A="abcde", X="acdeabdeabde" and Y="acde". Right : The visible history for $A|XY$ and the center M obtained from this alignment.

given column, then we assume them to be of the same size. This eliminates the case of a column of the form $[a,x,y]$, as this would require two reversals of different sizes.

$C(i,j,k)$ is the minimum over all the computed costs.

1. $[a,a,a]$: All matches.
$$M(i,j,k) = \begin{cases} C[i-1,j-1,k-1] & \text{if } A[i]=X[j]=Y[k] \\ +\infty & \text{otherwise} \end{cases}$$

2. $[a,x,x]$: Reversal in both X and Y (*i.e.* in M).
$$R_{XY}(i,j,k) = \begin{cases} min_{m\in E}(C[i-m,j-m,k-m]+c(R(m))) & \text{if } E\neq\emptyset \\ +\infty & \text{otherwise} \end{cases}$$
where E is the set $\{e_1,e_2,\ldots,e_l\}$ of maximum cardinality such that $A[i-e_p+1,i]$ is the reverse of both $X[j-e_p+1,j]$ and $Y[k-e_p+1,k]$ for all $1\leq p\leq l$.

3. $[a,x,a]$: Reversal in X. (The case $[a,a,y]$ is treated similarly)
$$R_X(i,j,k) = \begin{cases} min_{m\in E}(C[i-m,j-m,k-m]+c(R(m))) & \text{if } E\neq\emptyset \\ +\infty & \text{otherwise} \end{cases}$$
where E is the set $\{e_1,e_2,\ldots,e_l\}$ of maximum cardinality such that $A[i-e_p+1,i]=Y[k-e_p+1,k]$ and $A[i-e_p+1,i]$ is the reverse of $X[j-e_p+1,j]$ for all $1\leq p\leq l$.

4. $[-,x,x]$: Duplication in both X and Y (*i.e.* in M)

   $$D_{XY}(i,j,k) = \begin{cases} min_{1\leq m\leq l+1}(C[i,j-m,k-m]+c(D(m))) & \text{if } X[j]=Y[k] \\ +\infty & \text{otherwise} \end{cases}$$

   where l is the largest value such that $X[j-l,j]=Y[k-l,k]$ and $X[j-l,j]$ has an occurrence in A.

5. $[a,x,-]$: Reversal in both X and Y, and loss in Y. (The case $[a,-,y]$ is treated similarly)

   $$R_{X/Y}(i,j,k) = \begin{cases} min_{m\in E}(C[i-m,j-m,k]+c(R(m))+c(L(m))) & \text{if } E\neq\emptyset \\ +\infty & \text{otherwise} \end{cases}$$

   where E is the set $\{e_1,e_2,\ldots,e_l\}$ of maximum cardinality such that $A[i-e_p+1,i]$ is the reverse of $X[j-e_p+1,j]$ for all $1\leq p\leq l$.

6. $[-,x,y]$: Duplication in both X and Y, and reversal in Y.

   $$DR_{X/Y}(i,j,k) = \begin{cases} min_{m\in E}(C[i,j-m,k-m]+c(D(m))+c(R(m))) & \text{if } E\neq\emptyset \\ +\infty & \text{otherwise} \end{cases}$$

   where E is the set $\{e_1,e_2,\ldots,e_l\}$ of maximum cardinality such that $X[j-e_p+1,j]$ is the reverse of $Y[k-e_p+1,k]$ for all $1\leq p\leq l$ and $X[j-e_p+1,j]$ has an occurrence in A.

   (similar formulae for $DR_{Y/X}(i,j,k)$)

7. $[a,-,a]$: Loss in X. (The case $[a,a,-]$ is treated similarly)

   $$L_X(i,j,k) = \begin{cases} min_{1\leq m\leq l+1}(C[i-m,j,k-m]+c(L(m))) & \text{if } A[i]=Y[k] \\ +\infty & \text{otherwise} \end{cases}$$

   where $A[i-l,i]$ is the longest suffixe of $A[1,i]$ such that $A[i-l,i]=Y[k-l,k]$.

8. $[a,-,-]$: Loss in both X and Y.

   $$L_{XY}(i,j,k) = min_{0\leq m\leq i-1}(C[m,j,k]+c(L(i-m)))$$

9. $[-,x,-]$: Duplication in X. (The case $[-,-,y]$ is treated similarly)

$$D_X(i,j,k) = \begin{cases} min_{1\leq m\leq l+1}(C[i,j-m,k]+c(D(m))) & \text{if } X[j] \text{ has an occurrence in } A, X \text{ or } Y \\ +\infty & \text{otherwise} \end{cases}$$

where l is the largest value such that $X[j-l,j]$ has an occurrence in A, X or Y.

After computing all the values leading to $C(|A|,|X|,|Y|)$, the labeled alignment $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$ obtained by a backtracking approach is not necessarily a feasible alignment as it may contain cycles. Notice that, since A is an ancestor of both X and Y, the target of an event cannot belong to A. Therefore only events with source and target in X or Y may belong to a cycle.

• STEP 2. RESOLVING CYCLES.

Let $\mathscr{O}_c = \{O_1, O_2, \ldots, O_h\}$ be a cycle of a labeled alignment $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$ output by the above algorithm.

Lemma 1 *Any event of $\mathscr{O}_c$ is a duplication event.*

Proof: Suppose the contrary and let O_p be an event which is not a duplication. Then, by definition, the target O_p^T of O_p overlaps the source of O_{p+1}^S of O_{p+1}. Clearly, O_p cannot be a loss as otherwise O_p^T is empty and cannot have a non-empty intersection with O_{p+1}^S. Therefore O_p should be a reversal. Assume w.l.o.g. that O_p^T is in Y and let $Y[q]$ be an element of both O_p^T and O_{p+1}^S. Let $X[r]$ be the character of X aligned with $Y[r]$ in $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$. Then $X[r]$ should be in the source of O_p and in the target of O_{p+1}. But this leads to an interpretation of the corresponding column of $\mathscr{L}(\overline{A},\overline{X},\overline{Y})$ with two events instead of one, which is in contradiction with the recurrences leading to a minimum number of events for each column. □

We resolve cycles as follows. Let $\mathscr{Z}$ be the set of all overlapping strings $\{Z_1, Z_2, \ldots, Z_h\}$ of $\mathscr{O}_c$. Let $\mathscr{E}_i = \{z_{i_1}, z_{i_2}, \cdots z_{i_l}\}$ be a set of substrings of $Z_{i(1\leq i\leq h)}$ of minimum cardinality such that $z_{i_1} z_{i_2} \cdots z_{i_l} = Z_i$ and $z_{i_k(1\leq k\leq l)}$ has an occurrence in A. Let Z_t be the string for which $|\mathscr{E}_t| = min(|\mathscr{E}_1|, |\mathscr{E}_2|, \ldots, |\mathscr{E}_h|)$. Assume w.l.o.g. that Z_t is a substring of X. Then

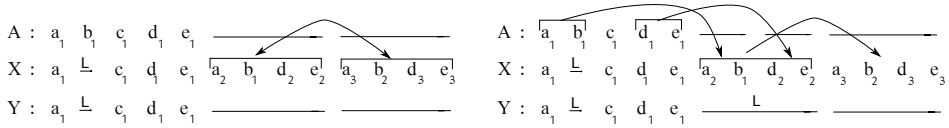
Z_t in $\mathscr{L}(\overline{X},\overline{Y})$ is covered by a loss in Y, and each substring of Z_t in $\mathscr{L}(\overline{A},\overline{X})$ is covered by a duplication in X (source in A) (see Figure 7.2 for details).

Complexity : For simplicity, assume that $|A| = |X| = |Y| = n$. From the recurrences detailed above, each $C(i,j,k)$ can be computed in linear time, leading to an $O(n^4)$ worst-time complexity for Step 1. Now, the complexity of Step 2 depends on the complexity for finding all cycles and resolving them. As cycles can only involve strings from X and Y, the problem reduces to the case of cycle-resolution for a pair-wise alignment, which has been shown quadratic (submitted journal version of [30]). This leads to a worst-time complexity of $O(n^4)$ for the whole algorithm.

7.5 Experimental Results

We call multiOrthoAlign our algorithm for the phylogenetic alignment problem based on the steinerization approach described in Section 7.4.3 and using our 3-star algorithm for the iteration step.

In this section, we compare multiOrthoAlign with DupLoCut [13], on simulated and real-world instances. DupLoCut is an "almost" exact heuristic based on linear programming. For the sake of comparison with DupLoCut [13], we consider a model restricted to



Labeled alignment with cycle : 2 duplications (cycle of duplications) and 1 loss in both X and Y

Feasible labeled alignment after resolving cycle : 3 duplications, 1 loss in both X and Y and 1 loss in Y.

Figure 7.2 : Two different labeling for the alignment of strings A="abcde", X="acdeabdeabde" and Y="acde". Losses are denoted by "L" and duplications by arrows from source (indicated by bracket) to target. In the left labeling, "$a_2b_1d_2e_2$" is interpreted as the target of a duplication. In the right one, it is interpreted in $\mathscr{L}(\overline{X},\overline{Y})$ as a loss, and in $\mathscr{L}(\overline{A},\overline{X})$ as 2 duplications.

duplications and single gene losses. Indeed, DupLoCut is restricted to this evolutionary model. Moreover, we consider the default cost of one for each event.

7.5.1 Simulations

We generate phylogenetic trees with 3 extant genomes. The genome at the root is generated in 2 steps. First, a random sequence R of length n on an alphabet of size σ is generated. Then, l moves (duplications and single gene losses) are applied to R where duplication length follows the geometric distribution of parameter 0.5. All other genomes along the tree are generated by applying l moves to their direct ancestor.

Execution time : We compare the running-time of our 3-star algorithm with that used in DupLoCut for the reoptimization steps. Running times were recorded using a 8-core Intel(R) 3.6 GHZ processor, with 16 GiB of memory. Table 7.I gives average running times after one round (iteration) of reoptimization for simulations generated with three choices of parameters n, σ and l. Although multiOrthoAlign's running time increases slightly with increasing values of n, σ and l, it is still within a few minutes for $n = 250$. In comparison, the same data took more than 6 hours to be processed by DupLoCut.

Accuracy : In order to test the performance of multiOrthoAlign in terms of accuracy, we used two measures : $Error = \frac{Inf-Opt}{Inf}$ where Inf is the number of events inferred

Average running times in minutes after one round of reoptimization		
Parameters (n, σ=n/10, l=n/3)	multiOrthoAlign	DupLoCut
(150,15,50)	0.33	48.93
(200,20,67)	0.90	110.12 ($\simeq$ 2 hours)
(250,25,84)	3.07	370.60 ($>$ 6 hours)

Tableau 7.I : Running times comparison between multiOrthoAlign and DupLoCut on simulated triplet phylogenies after one round of reoptimization. Times are averaged over 50 simulations for the first choice, and 10 simulations for the second and the last one. Average running times are reported in minutes.

by multiOrthoAlign and Opt is the "almost optimal" number of events obtained by running DupLoCut ; $Accuracy = \frac{NbOpt}{Total}$, where $NbOpt$ is the number of simulations among $Total$ (number of all simulations) for which multiOrthoAlign returns the same number of events as DupLoCut.

The same algorithm (2-SPP [148]) was used for the initialization step of both multiOrthoAlign and DupLoCut. Figure 7.3 gives results for different choices of the parameter l. With ratios $\sigma/n = 1/2$ and $l/n = 1/20$, multiOrthoAlign returns the same cost as DupLoCut for more than 96% of the simulations. This accuracy rate remains stable for decreasing alphabet size (results not shown), *i.e.*, for increasing number of gene copies, but decreases quickly as the number l of moves increases (left diagram of figure 7.3). However, the average $Error$ remains lower than 0.008 (right diagram of figure 7.3).

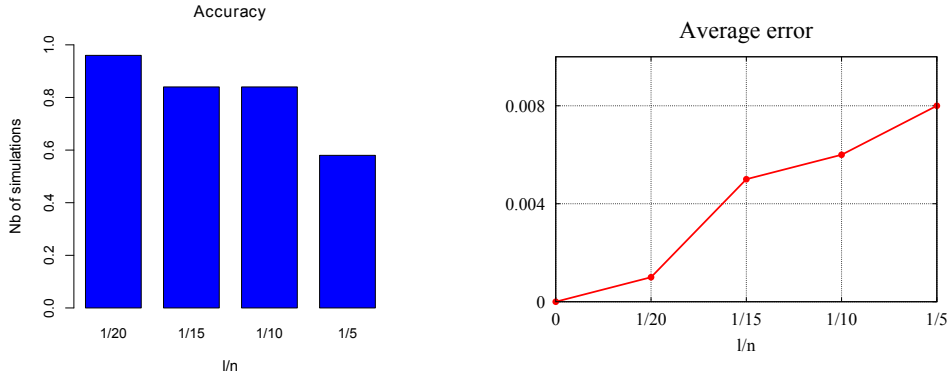


Figure 7.3 : The genome length is fixed to $n = 100$ and the alphabet size to $\sigma = n/2$; diagrams are obtained by varying the number of moves l (x-axis is l/n) ; results are averaged over 50 simulations. Left : Accuracy of multiOrthoAlign compared with DupLoCut. Right : the average $Error$.

In order to test the algorithm on larger trees, we generated a phylogenetic tree with 100 extant genomes. The genomes along the tree were generated as described above for triplet phylogenies, with parameters $n = 100$, $\sigma = 50$ and $l = 5$. Figure 7.4 illustrates the total cost of the tree (number of duplication/single gene loss events) obtained after each iteration of multiOrthoAlign (blue line) and DupLoCut (red line). After the initialization

step (iteration 0), the total cost obtained by multiOrthoAlign is 1632. After 6 rounds of reoptimization, the two programs converge to a local minimum (no improvement can be made), with a total cost of 1100 for multiOrthoAlign and of 1124 for DupLoCut. Our cost is always slightly better in this case. Notice that, although DupLoCut is "almost" exact for the median problem, the whole steinerization procedure does not guarantee any optimality result.

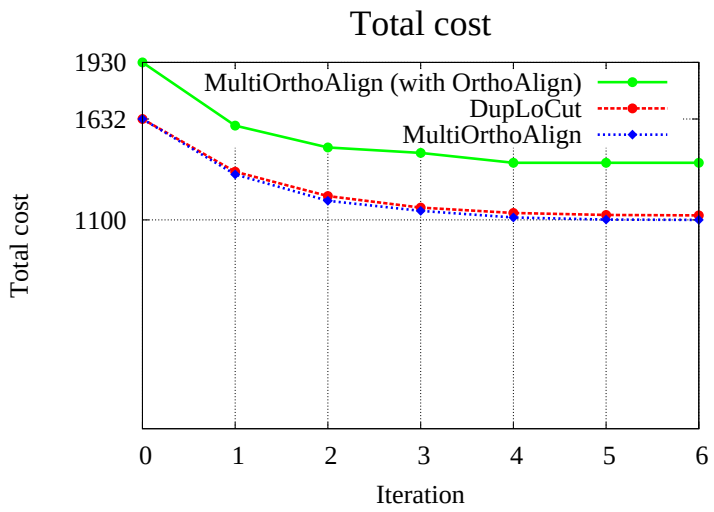


Figure 7.4 : Total cost obtained by multiOrthoAlign versus the one obtained by DupLoCut. Blue refers to the cost obtained by multiOrthoAlign when we used the 2-SPP algorithm for the initialization step, green to the cost obtained by multiOrthoAlign when we used OrthoAlign for the initialization step, and red to the cost obtained by DupLoCut.

Using OrthoAlign instead of the 2-SPP algorithm for the initialization step would be something natural to do for reducing the running time of the whole procedure. However, as illustrated in Figure 7.4 (green line), the initial assignment obtained with OrthoAlign in this case leads to a cost of 1930 which is far from the best solution found. Notice that 2-SPP is an exact algorithm for pair-wise alignment and OrthoAlign is a heuristic which does not guarantee the optimal result. multiOrthoAlign converge to a local minimum of 1401 events after 4 rounds of reoptimization.

7.5.2 Real data

We also compared the two approaches on the set of real-world instances used in [13]. The set contains the stable RNA genes of 12 *Bacillus* strains of four species (amyloliquefaciens, subtilis, thuringiensis, and cereus). The phylogeny shown in Figure 7.5 is taken from the webpage (http ://ccb.jhu.edu/software/duplocut).

Using 2-SPP for the initialization step, multiOrthoAlign leads to a cost of 136 after the initialization step, and converges to a local minimum of 123 events after 2 rounds of reoptimization. As for DupLoCut, it converges to a local minimum of 120 events after 5 rounds of reoptimization. However, using OrthoAlign instead of 2-SPP for the initialization step, multiOrthoAlign leads to a cost of 131 after the initialization step, which is not refined by subsequent iterations. It therefore appears that 2-SPP is a more appropriate initialization procedure than OrthoAlign.

7.6 Conclusion

We have developed multiOrthoAlign, a phylogenetic alignment algorithm for a genome-wide evolutionary model involving duplications, losses and rearrangements. It uses a generalization of OrthoAlign [243], a recently developed pair-wise alignment algorithm, to the median of three genomes. Our algorithm for the median problem is a heuristic that does not guarantee any optimality result. Compared with DupLoCut, the most

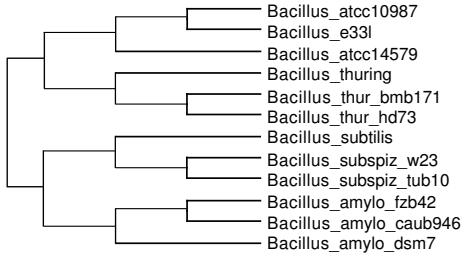


Figure 7.5 : Phylogenetic tree of the 12 *Bacillus* strains taken from the webpage (http ://ccb.jhu.edu/software/duplocut).

closely related existing algorithm, multiOrthoAlign exhibits similar results but is much faster. The method can be easily extended to other content-modifying and rearrangement operations such a substitutions, insertions, tandem duplications or inverted duplications. However, the more operations we add, the more challenging is the problem of finding appropriate costs for operations, and appropriate criteria to deal with the non-uniqueness of solutions.

CONCLUSION

Dans cette thèse, de nouvelles méthodes algorithmiques permettant l'étude de l'évolution de génomes ont été présentées. Plus particulièrement, nous avons développé des heuristiques, basées sur la programmation dynamique, pour le problème d'alignement de génomes représentés par des ordres de gènes, d'inférence d'ordres de gènes dans les génomes ancestraux et d'histoires évolutives dans un arbre phylogénétique donné.

L'heuristique DLAlign de complexité quadratique, pour le problème d'alignement de deux ordres de gènes par duplications et pertes, a été prouvée très efficace, en terme de solutions fournies, et très rapide par rapport à l'algorithme exact de complexité exponentielle [149], sur des données simulées. Ceci rend possible l'application de notre heuristique à des données biologiques réelles de grande taille. Nous avons également prouvé la APX-difficulté du problème IMA d'interprétation minimale d'un alignement de deux génomes par duplications et pertes. Ce problème est APX-difficile même dans le cas où le nombre de copies de chaque gène ne dépasse pas 5. En revanche, le problème peut être résolu en temps linéaire dans le cas où le nombre de copies de chaque gène est au plus 2.

Il serait intéressant d'étudier la complexité du problème IMA dans le cas où le nombre de copies de chaque gène est au plus 3 ou 4. Peut-on prouver la APX-difficulté de ce problème dans ce cas ? Si oui, peut-on proposer un algorithme exact pour la résolution du problème RMA, qui consiste à corriger de manière optimale l'alignement interprété non-réalisable ? Nos premières analyses semblent indiquer qu'une solution polynomiale existe pour ce problème. En effet, il est possible de représenter les duplications d'une interprétation donnée sous forme de graphe plus complexe, dans lequel toutes les occurrences d'un même gène appartiennent au même cycle de duplications et par conséquent, deux cycles quelconques du graphe ne partagent pas la même arête. Le problème consiste donc à simplement supprimer le minimum d'arêtes de chaque cycle.

Il serait ensuite intéressant d'étudier la complexité du problème IMA dans le cas général de coût quelconque des opérations. Rappelons que la complexité du problème IMA n'a été jusqu'à présent étudiée que dans le cas du coût unitaire.

DLAlign se généralise aisément aux opérations de réarrangement. Cette généralisation a fait l'objet d'une deuxième heuristique présentée dans cette thèse, OrthoAlign. Il serait intéressant d'améliorer OrthoAlign et ainsi de proposer un meilleur algorithme en terme de solutions fournies. En effet, dans certains cas, la solution optimale fait partie des solutions possibles fournies par l'heuristique. Il faudrait donc étendre la méthode pour permettre une exploration plus large de l'espace des solutions. Nous pourrions, par exemple, ajouter des contraintes afin de pouvoir déterminer une telle solution.

La troisième heuristique proposée, MultiOrthoAlign, est une généralisation de OrthoAlign à l'alignement de trois ordres de gènes. Elle permet d'étendre l'inférence de génomes ancestraux de cerises à une phylogénie. Il serait également souhaitable d'étendre le modèle évolutif à d'autres opérations comme le transfert horizontal. Le transfert de matériel génétique d'une espèce à une autre, joue en effet, un grand rôle dans la diversification des bactéries. Il est donc très important de pouvoir en tenir compte dans l'étude de l'évolution des génomes, en plus des événements de duplication, perte et de réarrangement. Un transfert horizontal peut être modélisé par une duplication dont la source et la cible appartiennent à deux génomes différents et dont la source n'a pas d'occurrence dans l'ancêtre des deux génomes. Cependant, dans certains cas, un transfert horizontal a le même effet sur l'alignement de deux génomes qu'une duplication suivie d'une perte. Utiliser uniquement notre approche d'alignement ne permet donc pas d'identifier spécifiquement les transferts horizontaux. Cette limitation réside dans le fait que la comparaison de génomes, basée uniquement sur les ordres de gènes, ne permet pas, à elle seule, d'inférer les transferts horizontaux. Il est donc important d'explorer les séquences de nucléotides des génomes en question et de déterminer le contenu en G+C (Guanine et Cytosine) qui constitue une source importante pour l'inférence de transfert

horizontal [172].

La limite la plus importante de notre modèle évolutif est l'hypothèse de visibilité des événements. Cela limite l'applicabilité de nos méthodes algorithmiques, et ne permet pas d'analyser des génomes trop éloignés évolutivement. Il serait intéressant de modifier nos approches d'alignement global en approches d'alignement local afin d'identifier des régions synténiques entre les génomes des espèces comparées. Ces régions correspondent à des segments chromosomiques dont le contenu en gènes est conservé entre les génomes. Une fois ces régions synténiques identifiées, il est possible de reconstruire directement les génomes ancestraux sans prédiction des scénarios qui les mènent aux génomes modernes. Le résultat de la comparaison ne sera pas toujours le génome entier, mais plutôt un ensemble de segments de chromosomes appelés *régions ancestrales contigües* RAC (*Contiguous Ancestral Regions* en anglais). La plupart des méthodes de reconstruction de génomes ancestraux, basées sur cette approche, conduisent à des RAC ne contenant qu'une seule copie de chaque gène. Notre approche d'alignement permettrait de supprimer cette limitation, c'est-à-dire de permettre l'inférence de RAC ayant plusieurs copies d'un gène.

Nous n'avons considéré dans cette thèse que des génomes dont l'ordre des gènes est complètement connu. Or, les méthodes de séquençage de nouvelle génération ont beaucoup progressé durant la dernière décennie, laissant accroître le nombre de génomes qualifiés de *partiellement assemblés* (ou "draft genomes" en anglais), comme ceux de vertébrés, étant donné que le coût de la production de ces génomes est en baisse constante. Ces génomes sont représentés par une collection de segments de gènes (nommés contigs), dont l'ordre et l'orientation sont inconnus. Notons que dans le cadre de la comparaison d'ordres de gènes, il est nécessaire de connaître l'ordre et l'orientation des segments de ces génomes. À cet effet, un problème algorithmique, connu sous le nom de *Block Ordering Problem*, a été introduit [129], et consiste à réarranger les segments de deux génomes partiellement assemblés, de sorte que la distance entre les génomes

résultants (génomes *complètement assemblés*) soit minimale. La distance d'inversion a d'abord été considérée pour ce problème [129] puis, la distance d'inversion pondérée et la distance d'échange de blocs ont été considérées pour une variante de ce problème dans laquelle seulement l'un des deux génomes est partiellement assemblé [73, 178]. Pour cette variante du problème, il serait intéressant de développer un algorithme qui considère d'autres opérations, comme la transposition. Notons que les algorithmes proposés pour ces deux problèmes considèrent des génomes unichromosomiques contenant exactement les mêmes gènes sans aucun gène dupliqué. Il serait intéressant d'élargir l'étude à des génomes ayant des contenus différents en gènes, et de permettre les gènes dupliqués. Dans le cadre d'un tel élargissement du modèle évolutif, rendant le problème significativement plus complexe, la contrainte d'opérations non-chevauchantes pourrait permettre de rendre le problème gérable, tout en étant plus proches des modèles biologiques dans certains cas. Il serait aussi intéressant de développer des heuristiques pour la reconstruction de séquences ancestrales dans un arbre phylogénétique donné dont les génomes représentés aux feuilles ne sont que partiellement assemblés.

Finalement, plusieurs auteurs commencent à utiliser des modèles évolutifs sous l'hypothèse d'opérations non-chevauchantes, pour rendre certains problèmes gérables. On signale, par exemple, les travaux de Cho et al. [75] qui considèrent le problème d'alignement de séquences de nucléotides par inversions et translocations. Ta et al. [236] considèrent le problème de calcul de distance d'inversion et de transposition. Zohora et Rahman [263] considèrent le problème d'existence d'ancêtre commun de deux gènes séparés par des inversions non-chevauchantes. En se basant sur cette hypothèse, ces auteurs relèvent également des problèmes ouverts qu'il serait intéressant d'étudier comme le problème de *séquence consensus* [12]. Bien que de plus en plus populaire, le modèle évolutif de génomes sous l'hypothèse d'opérations non-chevauchantes reste peu exploré. Cette thèse est une ouverture à ce domaine.

BIBLIOGRAPHIE

[1] M.I. Abouelhoda. Algorithms and software system for comparative genome analysis. *Phd thesis*, 2(1), 2005.

[2] M.I. Abouelhoda, S. Kurtz et E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[3] M.I. Abouelhoda, S. Kurtz et E. Ohlebusch. Coconut : an efficient system for the comparison and analysis of genomes. *BMC Bioinformatics*, 9(476), 2008.

[4] Z. Adam et D. Sankoff. The abcs of mgr with dcj. *Evol Bioinform Online*, 10(4): 69–74, 2008.

[5] M.A. Alekseyev. Multi-break rearrangements and breakpoint re-uses : From circular to linear genomes. *Journal of Computational Biology*, 15:1117–1131, 2008.

[6] M.A. Alekseyev et P.A. Pevzner. Whole genome duplications, multi-break rearrangements, and genome halving theorem. *in : Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 665–679, 2007.

[7] M.A. Alekseyev et P.A. Pevzner. Multi-break rearrangements and chromosomal evolution. *Theoretical Computer Science*, pages 193–202, 2008.

[8] S.F. Altschul, W. Gish, E. Miller, E.W. Meyers et D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[9] S.F. Altschul, T.L. Madden, A.A. Chäffer, J. Zhang, Z. Zhang, W. Miller et D.J. Lipman. Gapped blast and psi-blast : a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[10] A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena et U. Vishne. Pattern matching with address errors : Rearrangement distances. *J. Comput. System Sci*, 75:359–370, 2009.

[11] A. Amir, Y. Aumann, P. Indyk, A. Levy et E. Porat. Efficient computations of l_1 and l_∞đ rearrangement distances. *in String Processing and Information Retrieval, Lecture Notes in Comput. Sci*, 4726:39–49, 2007.

[12] A. Amir, H. Paryenty et L. Roditty. Approximations and partial solutions for the consensus sequence problem. *String Processing and Information Retrieval, LNCS*, 7024:168–173, 2011.

[13] S. Andreotti, K. Reinert et S. Canzar. The duplication-loss small phylogeny problem : From cherries to trees. *Journal of Computational Biology*, 20(9), 2013.

[14] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin et S. Vialette. A pseudo-boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes. *In Proceedings of the 5th RECOMB Comparative Genomics Satellite Workshop (RECOMB-CG)*, 4751:16–29, 2007.

[15] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin et S. Vialette. On the approximability of comparing genomes with duplicates. *Journal of Graph Algorithms and Applications*, 13(1):19–53, 2008.

[16] S.V. Angiuoli et S.L Salzberg. Mugsy : fast multiple alignment of closely related whole genomes. *Bioinformatics*, 27:334–342, 2011.

[17] T.S. Arruda, U. Dias et Z. Dias. Heuristics for the sorting by length-weighted inversion problem. *BCB'13*, pages 498–507, 2013.

[18] T.S. Arruda, U. Dias et Z. Dias. Heuristics for the sorting by length-weighted inversions problem on signed permutations. *Algorithms for Computational Biology, LNCS*, 8542:59–70, 2014.

[19] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela et M. Protasi. Complexity and approximation : Combinatorial optimization problems and their approximability properties. page 27, 1999.

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

[21] M. Bader. On reversal and transposition medians. *World Academy of Science, Engineering and Technology*, 3, 2009.

[22] M. Bader. The transposition median problem is NP-complete. *Theoretical Computer Science*, 412:1099–1110, 2011.

[23] M. Bader, M.I. Abouelhoda et E. Ohlebusch. A fast algorithm for the multiple genome rearrangement problem with weighted reversals and transpositions. *BMC Bioinformatics*, 9 :516, 2008.

[24] M. Bader et E. Ohlebusch. Sorting by weighted reversals, transpositions, and inverted transpositions. *J Comput Biol*, 14(5):615–636, 2007.

[25] V. Bafna, B.O. Narayanan et R. Ravi. Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). *Discrete Applied Mathematics*, 71(1-3):41–53, 1996.

[26] V. Bafna et P. A. Pevzner. Sorting permutations by transpositions. *in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 614–623, 1995.

[27] V. Bafna et P. A. Pevzner. Sorting permutations by transpositions. *SIAM J. Discrete Math*, 11:224–240, 1998.

[28] S. Batzoglou, L. Pachter, J.P. Mesirov, B. Berger et E.S. Lander. Human and mouse gene structure : comparative analysis and application to exon prediction. *Genome Research*, 10(7):950, 2000.

[29] C. Baudet, U. Dias et Z. Dias. Length and symmetry on the sorting by weighted inversions problem. *Advances in Bioinformatics and Computational Biology, LNCS*, 8826:99–106, 2014.

[30] B. Benzaid, R. Dondi et N. El-Mabrouk. Duplication-loss genome alignment : Complexity and algorithm. Dans *LNCS*, volume 7810 de *Language and Automata Theory and Applications, (LATA)*, pages 116-127, 2013.

[31] B. Benzaid, R. Dondi et N. El-Mabrouk. Genome alignment with duplication and loss : Complexity and algorithms. *Article soumis au journal IEEE/ACM Transactions on Computational Biology and Bioinformatics, TCBB*, 2015.

[32] B. Benzaid et N. El-Mabrouk. Gene order alignment on trees with multiorthoalign. *BMC-Genomics, special issue for RECOMB-CG*, 15(Suppl 6) :S5, 2014.

[33] A. Bergeron, C. Chauve et Y. Gingras. Formal models of gene clusters. Dans I. Mandoiu et A. Zelikovsky, éditeurs, *Bioinformatics algorithms : techniques and applications*, chaptitre 8. Wiley, 2008.

[34] A. Bergeron, P. Medvedev et J. Stoye. Rearrangement models and single-cut operations. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 17:1213–1225, 2010.

[35] A. Bergeron, J. Mixtacki et J. Stoye. On sorting by translocations. *Journal of Computational Biology*, 13(2):567–578, 2006.

[36] A. Bergeron, J. Mixtacki et J. Stoye. A unifying view of genome rearrangements. *In Proceedings of the 6th International Workshop on Algorithms in Bioinformatics (WABI06), LNCS*, 4175:163–173, 2006.

[37] A. Bergeron et J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. 13:1340–1354, 2003.

[38] P. Berman, S. Hannanhalli et M. Karpinski. 1.375-approximation algorithm for sorting by reversals. *In Proc. of 10th Eurpean Symposium on Algorithms (ESA 02)*, pages 200–210, 2002.

[39] M. Blanchette. Computation and analysis of genomic multi-sequence alignments. *Annu Rev Genomics Hum Genet*, 8:193–213, 2007.

[40] M. Blanchette, G. Bourque et D. Sankoff. Breakpoint phylogenies. *Genome Inform Ser Workshop Genome Inform*, 8:25–34, 1997.

[41] M. Blanchette, W. Kent, C. Riemer, L. Elnitsk, A. Smit, K. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson et E. Green. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708, 2004.

[42] M. Blanchette, T. Kunisawa et David Sankoff. Parametric genome rearrangement. *Gene*, 172:GC11–GC17, 1996.

[43] G. Blin, C. Chauve et G. Fertin. The breakpoint distance for signed sequences. *In Proceedings of the 1st Algorithms and Computational Methods for Biochemical and Evolutionary Networks (CompBioNets)*, pages 3–16, 2004.

[44] G. Blin, C. Chauve, G. Fertin, R. Rizzi et S. Vialette. Comparing genomes with duplications : a computational complexity point of view. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(4):523–534, 2007.

[45] G. Blin, G. Fertin, F. Sikora et S. Vialette. The exemplar breakpoint distance for non-trivial genomes cannot be approximated. *WALCOM, LNCS*, 5431:357–368, 2009.

[46] D. Bongartz. Algorithmic aspects of some combinatorial problems in bioinformatics. *Ph.D. thesis, Fakultat fur Mathematik, Informatik und Naturwissenschaften, RWTH Aachen, Aachen, Germany*, 2006.

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

[48] S. Boyd et M. Haghighi. Mixed and circular multichromosomal genomic median problem. *SIAM Journal of Discrete Mathematic*, 27(1):63–74, 2013.

[49] M.D.V. Braga, C. Chauve, D. Doerr, K. Jahn, J. Stoye, A. Thévenin et R. Wittler. *Models and algorithms for genome evolution*, chaptitre The potential of family-free genome comparison. Springer, 2013.

[50] M.D.V. Braga, R. Machado, L.C. Ribeiro et J. Stoye. Genomic distance under gene substitutions. *BMC Bioinformatics*, 12, 2011.

[51] M.D.V. Braga et J. Stoye. Restricted dcj-indel model revisited. *Advances in Bioinformatics and Computational Biology, LNCS*, 8213:36–46, 2013.

[52] M.D.V. Braga et J. Stoye. Sorting linear genomes with rearrangements and indels. *TCBB*, 12, 2015.

[53] M.D.V. Braga, E. Willing et J. Stoye. Genomic distance with dcj and indels. *Algorithms in Bioinformatics, LNCS*, 6293:90–101, 2010.

[54] M.D.V. Braga, E. Willing et J. Stoye. On sorting genomes with dcj and indels. *Comparative Genomics, LNCS*, 6398:62–73, 2010.

[55] M.D.V. Braga, E. Willing et J. Stoye. Double cut and join with insertions and deletions. *J. Comput. Biol*, 18:1167–1184, 2011.

[56] N. Bray, I. Dubchak et L. Pachter. Avid : A global alignment program. *Genome Research*, 13(1):97–102, 2003.

[57] M. Brudno, C.B. Do, G.M. Cooper, M.F. Kim, E. Davydov, E.D. Green, A. Sidow et S. Batzoglou. Lagan and multi-lagan : efficient tools for large-scale multiple alignment of genomic dna. *Genome Research*, 13(4):721–731, 2003.

[58] M. Brudno, C.B. Do, G.M. Cooper, M.F. Kim, E. Davydov, E.D. Green, A. Sidow et S. Batzoglou. LAGAN and Multi-LAGAN : efficient tools for large-scale multiple alignment of genomic dna. *Genome Research*, 13(4):721–731, 2003.

[59] M. Brudno, S. Malde, A. Poliakov, C.B. Do, O. Couronne, I. Dubchak et S. Batzoglou. Glocal alignment : finding rearrangements during alignment. *Bioinformatics*, 19(S1):i54–62, 2003.

[60] D. Bryant. The complexity of the breakpoint median problem. *In Tech. Rep. CRM-2579. Centre de recherches mathématiques, Université de Montréal*, 1998.

[61] D. Bryant. The complexity of calculating exemplar distances. *Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, 4751:207–212, 2000.

[62] D. Bryant. A lower bound for the breakpoint phylogeny problem. *J. Discrete Algorithms*, 2(2):229–255, 2004.

[63] L. Bulteau, G. Fertin et I. Rusu. Pancake flipping is hard. *In Proc. 37th International Symposium on Mathematical Foundations of Computer Science, Bratislava, Slovakia, LNCS*, 7464:247–258, 2012.

[64] L. Bulteau, G Fertin et I. Rusu. Sorting by transpositions is difficult. *SIAM J. Discrete Math*, 26(3):1148–1180, 2012.

[65] L. Bulteau et M. Jiang. Inapproximability of (1,2)-exemplar distance. *Bioinformatics Research and Applications, LNCS*, 7292:13–23, 2012.

[66] A. Caprara. Sorting by reversals is difficult. *RECOMB '97 Proceedings of the first annual international conference on Computational molecular biology*, pages 75–83, 1997.

[67] A. Caprara. Additive bounding, worst-case analysis, and the breakpoint median problem. *SIAM Journal on Optimization*, 13(2):508–519, 2002.

[68] A. Caprara. The reversal median problem. *INFORMS Journal on Computing*, 15: 93–113, 2003.

[69] H. Carillo et D.J. Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 88:1073–1082, 1988.

[70] S. Carroll, A. Griffiths, S. Wessler et R. Lewontin. Introduction à l'analyse génétique. 5ed, 2010.

[71] X. Chen. On sorting permutations by double-cut-and-joins. *Computing and Combinatorics, LNCS*, 6196:439–448, 2010.

[72] X. Chen et M. Tompa. Comparative assessment of methods for aligning multiple genome sequences. *Nat Biotechnol*, 28(6):567–572, 2010.

[73] K-T. Chena, C-L. Lia, H-T. Chiub et C.L. Lua. An efficient algorithm for one-sided block ordering problem under block-interchange distance. *Theoretical Computer Science*, 609(2):296–305, 2016.

[74] B. Chitturi et I. H. Sudborough. Bounding prefix transposition distance for strings and permutations. *Theoret. Comput. Sci*, 421:15–24, 2012.

[75] D-J. Cho, Y-S. Han et H. Kim. Alignment with non-overlapping inversions and translocations on two strings. *Theoretical Computer Science*, 575:90–101, 2015.

[76] D. A. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996.

[77] D. A. Christie et R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM J. Discrete Math*, 14:193–206, 2001.

[78] D.A. Christie. Genome rearrangement problems. *Ph.D. thesis, Department of Computer Science, University of Glasgow, Glasgow, Scotland*, 1998.

[79] T.C. Chu, T. Liu, D.T. Lee, G.C. Lee et A.C. Shih. Gr-aligner : an algorithm for aligning pairwise genomic sequences containing rearrangement events. *Discrete Applied Mathematics*, 25(17):2188–93, 2009.

[80] P.E.C. Compeau. A simplified view of dcj-indel distance. *WABI, LNBI*, 7534: 365–377, 2012.

[81] T.H. Cormen, C.E. Leiserson, R.L. Rivest et C. Stein. Introduction to algorithms. *MIT Press, 2nd edition*, 2001.

[82] G. Cormode et S. Muthukrishnan. The string edit distance matching problem with moves. *in Proceedings of the Thirteenth Annual Symposium on Discrete Algorithms*, pages 667–676, 2002.

[83] P. Crescenzi et L. Trevisan. On approximation scheme preserving reducibility and its applications. *Theory of Computing Systems*, 33(1):1–16, 2000.

[84] P.H. da Silva, R. Machado, S. Dantas et M.D.V. Braga. Restricted dcj-indel model : sorting linear genomes with dcj and indels. *BMC Bioinformatics*, 13(19), 2012.

[85] A.C Darling, B. Mau, F.R. Blattner et N.T. Perna. Mauve : Multiple alignment of conserved genomic sequence with rearrangements. *Genome Research*, 14(7): 1394–1403, 2004.

[86] A.E. Darling, B. Mau, F.R. Blattner et N.T. Perna. Gril : Genome rearrangement and inversion locator. *Bioinformatics*, 20:122–124, 2003.

[87] A.E. Darling, B. Mau et N.T. Perna. progressiveMauve : multiple genome alignment with gene gain, loss and rearrangement. *PLoS One*, 5(6):e11147, 2010.

[88] M.O. Dayhoff, R.M. Schwartz et B.C. Orcutt. A model for evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.

[89] A.L Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. White et S.L Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[90] A.L. Delcher, A. Phillippy, J. Carlton et S.L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, 2002.

[91] C. Demetrescu et I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 86(3):129 – 136, 2003.

[92] J. Deogun, J. Yang et F. Ma. Emagen : An efficient approach to multiple whole genome alignment. *In Proceedings of the second Conference on Asia-Pacific bioinformatics*, 29:122, 2004.

[93] C.N Dewey, P.M Huggins, K. Woods, B. Sturmfels et L. Pachter. Parametric alignment of drosophila genomes. *PLOS : Computational Biology*, 2:e(73), 2006.

[94] U. Dias et Z. Dias. Heuristics for the transposition distance problem. *J. Bioinform. Comput. Biol*, 11:1350013, 2013.

[95] U. Dias, G.R. Galvao, C.N. Lintzmayer et Z. Dias. A general heuristic for genome rearrangement problems. *J. Bioinform. Comput. Biol*, 12:1450012, 2014.

[96] Z. Dias et U. Dias. Sorting by prefix reversals and prefix transpositions. *Discrete Applied Mathematics*, 181:78–89, 2015.

[97] Z. Dias et J. Meidanis. Sorting by prefix transpositions. *In Proceedings of the 9th International Symposium on String Processing and Information Retrieval, Lecture Notes in Comput. Sci*, 2476:65–76, 2002.

[98] C. Do, M. Mahabhashyam, M. Brudno et S. Batzoglou. Probcons : probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15(2):330, 2005.

[99] R. Dondi et N. El-Mabrouk. Aligning and labeling genomes under the duplication-loss model. Dans *LNCS*, volume 7921 de *Computability in Europe (CiE 2013)*, pages 97-107, 2013.

[100] A. Dress, G. Füllen et S. Perrey. A divide and conquer approach to multiple alignment. *Proc Int Conf Intell Syst Mol Biol*, 3:107–13, 1995.

[101] D. Durand et D. Sankoff. Testing for gene clusters. *Journal of Computational Biology*, 10:453-482, 2003.

[102] H. Dweighter. Problem e2569. *Amer, Math. Monthly*, 82:1010, 1975.

[103] P. Eades et X. Lin. A new heuristic for the feedback arc set problem. *Australian Journal of Combinatorics*, 12:15 – 26, 1995.

[104] P. Eades, X. Lin et W.F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319 – 323, 1993.

[105] R. Edgar. Muscle : a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):113, 2004.

[106] J.A. Eisen, J.F. Heidelberg, O. White et S.L. Salzberg. Evidence for symmetric chromosomal inversions around the replication origin in bacteria. *Genome Biology*, 1(6):1–9, 2000.

[107] N. El-Mabrouk. Sorting signed permutations by reversals and insertions/deletions of contiguous segments. *Journal of Discrete Algorithms*, 1:105–122, 2001.

[108] N. El-Mabrouk et D. Sankoff. The reconstruction of doubled genomes. *SIAM Journal on Computing*, 32(1):754–792, 2003.

[109] N. El-Mabrouk et D. Sankoff. Analysis of gene order evolution beyond single-copy genes. *Evolutionary Genomics : statistical and computational methods, Part II*, 855:397–429, 2012.

[110] I. Elias et T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biol. Bioinform*, 3:369–379, 2006.

[111] D. Eppstein, Z. Galil, R. Giancarlo et G.F. Italiano. Sparse dynamic programming i : Linear cost functions. *Journal of Association for Computing Machinery*, 39(3): 519–545, 1992.

[112] H. Eriksson, K. Eriksson, J. Karlander, L. J. Svensson et J. Wastlund. Sorting a bridge hand. *Discrete Math*, 241:289–300, 2001.

[113] P. Feijao et J. Meidanis. Scj : A variant of breakpoint distance for which sorting, genome median and genome halving problems are easy. *Algorithms in Bioinformatics, LNCS*, 5724:85–96, 2009.

[114] J. Felsenstein. Inferring phylogenies. *Sinauer Associates*, 2004.

[115] S. Felsner, R. Muller et L. Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32, 1995.

[116] J. Feng et D. Zhu. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Trans. Algorithms*, 3, 2007.

[117] D. Ferre, R. Roset, M. Huerta, J.E. Adsu ara, L. Rosello, M. Alba et X. Messeguer. Identification of patterns in biological sequences at the alggen server : Promo and malgen. *Nucleic Acids Research*, 31(13):3651–3653, 2003.

[118] G. Fertin, A. Labarre, I. Rusu, E. Tannier et S. Vialette. *Combinatorics of genome rearrangements*. The MIT Press, Cambridge, Massachusetts and London, England, 2009.

[119] P. Festa, P. M. Pardalos et M. G. C. Resende. Feedback set problems. *Handbook of Combinatorial Optimization*, 4:209 – 258, 1999.

[120] G. Fischer, E.P Rocha, F. Brunet, M. Vergassola et B. Dujon. Highly variable rates of genome rearrangements between hemiascomycetous yeast lineages. *PLoS Genet*, 2(3), 2006.

[121] J. Fischer et S.W. Ginzinger. A 2-approximation algorithm for sorting by prefix reversals. *Algorithms - ESA. Lecture Notes in Computer Science*, 3669:415–425, 2005.

[122] M.M. Flood. Exact and heuristic algorithms for the weighted feedback arc set problem. a special case of the skew-symmetric quadratic assignment problem. *Networks*, 20:1 – 23, 1990.

[123] M. Benot Gagné et S. Hamel. A new and faster method of sorting by transpositions. *In Combinatorial Pattern Matching, Lecture Notes in Computer Science*, 4580:131–141, 2007.

[124] Y. Gagnon, M. Blanchette et N. El-Mabrouk. A flexible ancestral genome reconstruction method based on gapped adjacencies. *BMC Bioinformatics*, 13, 2012.

[125] G.R. Galvao et Z. Dias. Approximation algorithms for sorting by signed short reversals. *BCB '14*, pages 360–369, 2014.

[126] G.R. Galvao, O. Lee et Z. Dias. Sorting signed permutations by short operations. *Algorithms for Molecular Biology*, 2015.

[127] M.R. Garey et D.S. Johnson. *Computers and Intractability : A Guide to Theory of NP-completeness*. W.H. Freeman, New York, 1997.

[128] H. Gates et C. H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979.

[129] É. Gaul et M. Blanchette. Ordering partially assembled genomes using gene arrangements. *Comparative Genomics, LNCS*, 4205:113–128, 2006.

[130] C. Grasso et C. Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, 20 (10):1546–56, 2004.

[131] Q-P. Gu, S. Peng et Q.M. Chen. Sorting permutations and its applications in genome analysis. *Mathematical and Computational Biology, Lectures Math. Life Sci, AMS, Providence, RI*, 26:191–201, 1999.

[132] D. Gusfield. Algorithms on strings, trees and sequences. *Cambridge University Press*, 1997.

[133] S. A. Guyer, L. S. Heath et J. P. Vergara. Subsequence and run heuristics for sorting by transpositions. *Technical report, Virginia Polytechnic Institute and State University, Blacksburg, VA*, 1997.

[134] B.J. Haas, A.L. Delcher, J.R. Wortman et S.L. Salzberg. DAGchainer : a tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18): 3643-3646, 2004.

[135] M. Haghighi et S. Boyd. A fast method for large-scale multichromosomal breakpoint median problems. *Proceeding BCB '11 Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, pages 162–171, 2011.

[136] S. Hannenhalli. Polynomial algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996.

[137] S. Hannenhalli et P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). Dans *Proceedings of the IEEE 36th Annual Symposium on Foundations of Computer Science*, pages 581–592, 1995.

[138] S. Hannenhalli et P. A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Journal of the ACM*, 48: 1–27, 1999.

[139] S. Hannenhalli et P.A. Pevzner. Transforming cabbage into turnip : polynomial algorithm for sorting signed permutations by reversals. *In Proceedings of the 27th annual ACM symposium on Theory of computing*, pages 178–189, 1995.

[140] A T. Hartman et R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204:275–290, 2006.

[141] F. Farnoud (Hassanzadeh) et O. Milenkovic. Sorting of permutations by cost-constrained transpositions. *IEEE Transactions on Information Theory*, 58(1):3–23, 2012.

[142] S. Heber et J. Stoye. Finding all common intervals of k permutations. Dans A. Amir et G. M. Landau, éditeurs, *Combinatorial Pattern Matching. 12th Annual Symposium*, volume 2089 de *Lecture Notes in Computer Science*, pages 207– 218. Springer, 2001.

[143] JF Henikoff et S. Henikoff. Automated assembly of protein blocks for database searching. *Proceedings of the National Academy of Sciences of the USA*, 89: 10915–10919, 1992.

[144] A. Herencsar et B. Brejova. An improved algorithm for ancestral gene order reconstruction. *CEUR Workshop Proceedings*, 1214:46–53, 2014.

[145] M. H. Heydari et I. H. Sudborough. On the diameter of the pancake network. *J. of Algorithms*, 25:67–94, 1997.

[146] H.Koehler. A contraction algorithm for finding minimal feedback sets. Dans *ACSC 2005*, pages 165–174, 2005.

[147] M. Hohl, S. Kurtz et E. Ohlebusch. Efficient multiple genome alignment. *Bioinformatics*, 18(S1):S312–320, 2002.

[148] P. Holloway, K. Swenson, D. Ardell et N. El-Mabrouk. Ancestral genome organization : an alignment approach. *Journal of Computational Biology*, 20(4): 280-295, 2013.

[149] P. Holloway, K. M. Swenson, D. H. Ardell et N. El-Mabrouk. Evolution of genome organization by duplication and loss : An alignment approach. *B. Chor (Ed.) : RECOMB, LNBI*, 7262:94–112, 2012.

[150] W. Huang, D. Umbach et L. Li. Accurate anchoring alignment of divergent sequences. *Bioinformatics*, 22(1):29, 2005.

[151] Y.L Huang, C.C. Huang, C.Y. Tang et C.L Lu. An improved algorithm for sorting by block-interchanges based on permutation groups. *Information Processing Letters*, 110:345–350, 2010.

[152] G. Jacobson et K.-P Vo. Heaviest increasing/common subsequence problems. *In Proceedings of Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson*, 644:52–66, 1992.

[153] N. Jareborg, E. Birney et R. Durbin. Comparative analysis of noncoding regions of 77 orthologous mouse and human gene pairs. *Genome Research*, 9:815–824, 1999.

[154] M. Jiang. The zero exemplar distance problem. *Journal of Computational Biology*, 18(9):1077–1086, 2011.

[155] D. Joseph, J. Meidanis et P. Tiwari. Determining dna sequence similarity using maximum independent set algorithms for interval graphs. *Lecture Notes in Computer Science*, 621:326–337, 1992.

[156] H. Kaplan, R. Shamir et R.E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing*, 29(3):880–892, 1999.

[157] H. Kaplan et E. Verbin. Efficient data structures and a new randomized approach for sorting signed permutations by reversals. *Proceedings of the 14th Symposium on Combinatorial Pattern Matching. Lecture Notes in Computer Science*, 2676: 170–185, 2003.

[158] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[159] K. Katoh, K. Misawa, K. Kuma et T. Miyata. Mafft : a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Research*, 30(14):3059, 2002.

[160] J. Kececioglu et D. Sankoff. Exact and approximation algorithms for the inversion distance between two permutations. *In Proc. of 4th Ann. Symp. on Combinatorial Pattern Matching, LNCS*, 684:87–105, 1993.

[161] J.D. Kececioglu et R. Ravi. Of mice and men : Algorithms for evolutionary distances between genomes with translocation. *In : Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM Press*, pages 604–613, 1995.

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

[163] W. Kent, R. Baertsch, A. Hinrichs, W. Miller et D. Haussler. Evolution's cauldron : duplication, deletion, and rearrangement in the mouse and human genomes.

Proceedings of the National Academy of Sciences of the United States of America, 100(20):11484, 2003.

[164] W. Kent et A. Zahler. Conservation, regulation, synteny, and introns in a large-scale c. briggsae-c. elegans genomic alignment. *Genome Research*, 10(8):1115, 2000.

[165] P. Kolmana, et T. Waleń. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, 155(3):327–336, 2007.

[166] P. Kolmana, et T. Waleń. Reversal distance for strings with duplicates : Linear time approximation using hitting set. *Approximation and Online Algorithms. Lecture Notes in Computer Science*, 4368:279–289, 2007.

[167] J. Kovac, M.D.V Braga et J. Stoye. The problem of chromosome reincorporation in dcj sorting and halving. *Comparative Genomics, LNCS*, 6398:13–24, 2010.

[168] J. Kovac, B. Brejova et T. Vinar. A practical algorithm for ancestral rearrangement reconstruction. Dans *LNBI*, volume 6833 de *WABI*, pages 163- 174, 2011.

[169] J. Kovac, R. Warren, M.D.V Braga et J. Stoye. Restricted dcj model : Rearrangement problems with chromosome reincorporation. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 18:1231–1241, 2011.

[170] A. Labarre. New bounds and tractable instances for the transposition distance. *IEEE/ACM Trans. Comput. Biol. Bioinform*, 3:380–394, 2006.

[171] A. Labarre. Edit distances and factorisations of even permutations. *Algorithms - ESA, Lecture Notes in Computer Science.*, 5193:635–646, 2008.

[172] J.G. Lawrence et H. Ochman. Amelioration of bacterial genomes : rates of change and exchange. *J Mol Evol*, 44(4):383–397, 1997.

[173] C. Ledergerber et C. Dessimoz. Alignments with non-overlapping moves, inversions and tandem duplications in o(n4) time. *Computing and Combinatorics*, 4598:151–164, 2007.

[174] J.F. Lefebvre, N. El-Mabrouk, E.R.M. Tillier et D. Sankoff. Detection and validation of single gene inversions. *Bioinformatics*, 19:i190–i196, 2003.

[175] C. Lemaitre. Réarrangements chromosomiques dans les génomes de mammifères : caractérisation des points de cassure. *Phd*, 2009.

[176] R. Lenne, C. Solnon, T. Stutzle, E Tannier et M. Birattari. Reactive stochastic local search algorithms for the genomic median problem. *J. van Hemert and C. Cotta (Eds.) : EvoCOP, LNCS*, 4972:266–276, 2008.

[177] H. Levy et L. Low. A contraction algorithm for finding small cycle cutsets. *Journal of Algorithms*, 9:470 – 493, 1988.

[178] C-L. Li, K-T. Chen et C.L. Lu. Assembling contigs in draft genomes using reversals and block-interchanges. *Proceedings of the Third Annual RECOMB Satellite Workshop on Massively Parallel Sequencing (RECOMB-seq 2013)*, 2013.

[179] G. Li, X. Qi, X. Wang et B. Zhu. A linear-time algorithm for computing translocation distance between signed genomes. *Combinatorial Pattern Matching, LNCS*, 3109:323–332, 2004.

[180] Y.C. Lin, C.L. Lu, H.Y. Chang et C.Y. Tang. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *Journal of Computational Biology*, 12(1):102–112, 2005.

[181] C.N Lintzmayer et Z. Dias. On sorting of signed permutations by prefix and suffix reversals and transpositions. *Algorithms for Computational Biology, Lecture Notes in Computer Science*, 8542:146–157, 2014.

[182] C.N Lintzmayer et Z. Dias. Sorting permutations by prefix and suffix versions of reversals and transpositions. *LATIN 2014 : Theoretical Informatics Lecture Notes in Computer Science*, 8392:671–682, 2014.

[183] C.N Lintzmayer, G. Fertin et Z. Dias. Approximation algorithms for sorting by length-weighted prefix and suffix operations. *Theoretical Computer Science*, 593: 26–41, 2015.

[184] C.L. Lucchesi. *A minimax equality for directed graphs*. PhD thesis, University of Waterloo, 1966.

[185] B. Ma, J. Tromp et M. Li. Patternhunter : faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2003.

[186] M. Marron, K.M. Swenson et B.M.E. Moret. Genomic distances under deletions and insertions. *Theoretical Computer Science,*, 325:347–360, 2004.

[187] B.M.E Moret, A. Siepel, J. Tang et T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. *Algorithms in Bioinformatics*, pages 521–536, 2002.

[188] B.M.E Moret, L. Wang, T. Warnow et S. Wyman. New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17(S165), 2001.

[189] B. Morgenstern, K. Frech, A.W.M Dress et T. Werner. Dialign : finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998.

[190] G. Myers et W. Miller. Chaining multiple-alignment fragments in subquadratic time. *In Proceedings of the sixth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 38–47, 1995.

[191] S.B Needleman et C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[192] C.T. Nguyen, Y.C. Tay et L. Zhang. Divide-and-conquer approach for the exemplar breakpoint distance. *Bioinformatics*, 21:2171–2176, 2005.

[193] D.A Nix et M.B. Eisen. Gata : A graphic alignment tool for comparative sequence analysis. *BMC Bioinformatics*, 6, 2005.

[194] L. Noé et G. Kucherov. Yass : enhancing the sensitivity of dna similarity search. *Nucleic Acids Research*, 33(Web Server Issue) :W540, 2005.

[195] C. Notredame. Recent evolutions of multiple sequence alignment algorithms. *PLoS Computational Biology*, 3(8):330, 2007.

[196] C. Notredame, D. Higgins et J. Heringa. T-coffee : a novel method for fast and accurate multiple sequence alignment1. *Journal of Molecular Biology*, 302(1): 205–217, 2000.

[197] E. Ohlebusch, M.I. Abouelhoda et K. Hockel. A linear time algorithm for the inversion median problem in circular bacterial genomes. *Journal of Discrete Algorithms*, 5:637–646, 2007.

[198] O. O'Sullivan, K. Suhre, C. Abergel, D. Higgins et C. Notredame. 3dcoffee : combining protein sequences and structures within multiple sequence alignments. *Journal of Molecular Biology*, 340(2):385–395, 2004.

[199] I. Ovcharenko, G.G. Loots, B.M Giardine, H. Hou, J. Ma, R.C. Hardison, L. Stubbs et W. Miller. Multiple-sequence local alignment and visualization for studying function and evolution. *Genome Research*, 15:184, 2004.

[200] M. Ozery-Flato et R. Shamir. Two notes on genome rearrangement. *Journal of Bioinformatics and Computational Biology*, 1:71–94, 2003.

[201] M. Ozery-Flato et R. Shamir. An $o(n^{3/2}\sqrt{\log n})$ algorithm for sorting by reciprocal translocations. *Journal of Discrete Algorithms*, 9:344–357, 2011.

[202] C. H. Papadimitriou et M. Yannakakis. Optimization, approximation and complexity classes. *J. Comput. System Sci*, 43:425–440, 1991.

[203] C.H. Papadimitriou. Computational complexity. *Addison Wesley*, 1994.

[204] B. Paten, J. Herrero, K. Beal, S. Fitzgerald et E. Birney. Enredo and pecan : Genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Research*, 18(11):1814–1828, 2008.

[205] W.R. Pearson et D.J. Lipman. Improved tools for biological sequence comparison. Dans *In Proc Natl Acad Sci*, volume 85(8), pages 2444–2448, USA, 1988.

[206] I. Pe'er et R. Shamir. The median problems for breakpoints are np-complete. *Tech. Rep. TR98-071, Electronic Colloquium on Computational Complexity*, 1998.

[207] I. Pe'er et R. Shamir. Approximation algorithms for the median problem in the breakpoint model. *In D. Sankoff, J.H. Nadeau (Eds.), Comparative Genomics, Kluwer, Dordrecht*, pages 225–241, 2000.

[208] V.Y. Popov. On the diameter of the pancake network. *IAENG International Journal of Applied Mathematics*, 40(4), 2010.

[209] A. J. Radcliffe, A. D. Scott et E. L. Wilmer. Reversals and transpositions over finite alphabets. *SIAM J. Discrete Math*, 19:224–244, 2005.

[210] V. Rajan, A. Xu, Y. Lin, K. Swenson et B.M.E Moret. Heuristics for the inversion median problem. *BMC bioinformatics*, 11 (S30), 2010.

[211] B. Raphael, D. Zhi, H. Tang et P. Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14(11): 2336, 2004.

[212] J.L. Risler et D. Tagu. Bio-informatique - principes d'utilisation des outils. *Quae*, 2010.

[213] M. Roytberg, A. Ogurtsov, S. Shabalina et A. Kondrashov. A hierarchical approach to aligning collinear regions of genomes. *Bioinformatics*, 18(12):1673, 2002.

[214] I. Rusu. Log-lists and their applications to sorting by transpositions, reversals and block-interchanges. *arXiv :1507.01512*, 2015.

[215] J.W. Sahl, J.G. Caporaso, D.A. Rasko et P. Keim. The large-scale blast score ratio (ls-bsr) pipeline : a method to rapidly compare genetic content between bacterial genomes. *Peer J*, 2:e332, 2014.

[216] D. Sankof. Mechanisms of genome evolution : models and inference. *Bulletin of the International Statistical Institute*, 47:461–475, 1989.

[217] D. Sankof et M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570, 1998.

[218] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, pages 909–917, 1999.

[219] D. Sankoff et M. Blanchette. The median problem for breakpoints in comparative genomics. *Proceedings of the Third International Computing and Combinatorics Conference COCOON97*, pages 251–263, 1997.

[220] D. Sankoff, R. Cedergren et G. Lapalme. Frequency of insertion-deletion, transver- sion, and transition in the evolution of 5S ribosomal RNA. *Journal of Molecular Evolution*, 7:133-149, 1976.

[221] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B.F. Lang et R. Cedergren. Gene order comparisons for phylogenetic inference : evolution of the mitochondrial genome. *Proc Natl Acad Sci U S A*, 89(14):6575–6579, 1992.

[222] M. Schoniger et M. S. Waterman. A local algorithm for dna sequence alignment with inversions. *Bulletin of Mathematical Biology*, 54(4):521–536, 1992.

[223] S. Schwartz, W. Kent, A. Smit, Z. Zhang, R. Baertsch., R. Hardison., D. Haussler et W. Miller. Human-mouse alignments with BLASTZ. *Genome research*, 13: 103–107, 2003.

[224] S. Schwartz, Z. Zhang, K. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison et W. Miller. Pipmaker-a web server for aligning two genomic dna sequences. *Genome Research*, 10(4):577, 2000.

[225] M. Shao et Y. Lin. Approximating the edit distance for genomes with duplicate genes under dcj, insertion and deletion. *BMC Bioinformatics*, 2012.

[226] M. Shao, Y. Lin et B. Moret. An exact algorithm to compute the dcj distance for genomes with duplicate genes. *RECOMB, LNCS*, 8394:280–292, 2014.

[227] M. Shao et B.M.E. Moret. Comparing genomes with rearrangements and segmental duplications. *Bioinformatics*, 31:329–338, 2015.

[228] M. Shao et B.M.E Moret. A fast and exact algorithm for the exemplar breakpoint distance. *Proc. 19th Int'l Conf. on Research in Comput. Molecular Bio. RECOMB'15, LNCS*, 9029:309–322, 2015.

[229] D. Shapira et J. A. Storer. Edit distance with move operations. *in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci*, 2373:85–98, 2002.

[230] M. Sharmin, R. Yeasmin, M. Hasan, A. Rahman et M.S. Rahman. Pancake flipping with two spatulas. *Electron. Notes Discrete Math*, 36:231–238, 2010.

[231] A.C. Siepel et B.M.E. Moret. Finding an optimal inversion median : Experimental results. *In Proc. WABI*, pages 189–203, 2001.

[232] T.F. Smith et M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[233] K.M. Swenson, V. Rajan, Y. Lin et B.M.E. Moret. Sorting signed permutations by inversions in o(n log n) time. *Proceedings of the 13th Annual International Conference on Research in Computational Molecular Biology (RECOMB09). Lecture Notes in Computer Science*, 5541:386–399, 2009.

[234] F. Swidan, M. Bender, D. Ge, S. He, H. Hu et R. Pinter. Sorting by length-weighted reversals : Dealing with signs and circularity. *Combinatorial Pattern Matching LNCS*, 3109:32–46, 2004.

[235] F. Swidan, E.P.C. Rocha, M. Shmoish et R.Y. Pinter. An integrative method for accurate comparative genome mapping. *PLoS Comput Biology*, 2(8):e75, 2006.

[236] T.T. Ta, C-Y. Lin et C.L. Lu. An efficient algorithm for computing non-overlapping inversion and transposition distance. *The 32nd Workshop on Combinatorial Mathematics and Computation Theory*, 2015.

[237] J. Tang et B.M.E. Moret. Phylogenetic reconstruction from gene-rearrangement data with unequal gene content. *In Proceedings of Workshop on Algorithms and Data Structures (WADS03), LNCS*, 2748:37–46, 2003.

[238] E. Tannier, A. Bergeron et M.F. Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155:881–888, 2007.

[239] E. Tannier et M.F. Sagot. Sorting by reversals in subquadratic time. *Combinatorial Pattern Matching. Lecture Notes in Computer Science*, 3109:1–13, 2004.

[240] E. Tannier, C. Zheng et D. Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10, 2009.

[241] J. Thompson, D. Higgins et T. Gibson. Clustal w : improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22): 4673–4680, 1994.

[242] T. Treangen et X. Messeguer. M-gcat : interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. *BMC Bioinformatics*, 7(1):433, 2006.

[243] O. Tremblay-Savard, B. Benzaid, B. F. Lang et N. El-Mabrouk. Evolution of trna repertoires in bacillus inferred with orthoalign. *Molecular Biology and Evolution*, 32(6):1643–1656, 2015.

[244] R. Uricaru. Algorithmes de comparaison de génomes appliqués aux génomes bactériens. *Phd thesis*, 2010.

[245] A.F Vellozo, C.E.R Alves et A.P. do Lago. Alignment with non-overlapping inversions in o(n3)-time. *P. Bucher and B.M.E. Moret (Eds.) : WABI, LNBI*, 4175: 186–196, 2006.

[246] R. Vincent. Génétique moléculaire. *De Boeck*, 2007.

[247] V.Ramachandran. A minimax arc theorem for reducible flow graphs. *SIAM Journal on Discrete Mathematics*, 3(4):554 – 560, 1990.

[248] R.H. Waterston, E. Birney K. Lindblad-Toh, J. Rogers, J.F. Abril, R. Agarwala P. Agarwal, R. Ainscough, M. Alexandersson, P. An et al. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–562, 2002.

[249] S. Xiangqian, J. Xing, Z. Wang, L. Chen, M. Cui et B. Jiang. micrornas and cernas : Rna networks in pathogenesis of cancer. *Chin J Cancer Res*, 25(2):235–239, 2013.

[250] A.W. Xu. The median problems on linear multichromosomal genomes : Graph representation and fast exact solutions. *Journal of Computational Biology*, 17(9): 1195–1211, 2010.

[251] A.W. Xu et B.M.E. Moret. Gasts : Parsimony scoring under rearrangements. *Algorithms in Bioinformatics, LNCS*, 6833:351–363, 2011.

[252] A.W. Xu et D. Sankoff. Decompositions of multiple breakpoint graphs and rapid exact solutions. *K.A. Crandall and J. Lagergren (Eds.) : WABI, LNBI*, 5251:25–37, 2008.

[253] S. Yancopoulos, O. Attie et R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16): 3340–3346, 2005.

[254] S. Yancopoulos et R. Friedberg. Sorting genomes with insertions, deletions and duplications by dcj. *Comparative Genomics, LNCS*, 5267:170–183, 2008.

[255] Z. Yang et D. Sankoff. Natural parameter values for generalized gene adjacency. *Journal of Computational Biology*, 17:1113-1128, 2010.

[256] Z. Yin, J. Tang, S.W. Schaeffer et D.A. Bader. Streaming breakpoint graph analytics for accelerating and parallelizing the computation of dcj median of three genomes. *Procedia Computer Science*, 18:561–570, 2013.

[257] Z. Yin, J. Tang, S.W. Schaeffer et D.A. Bader. A lin-kernighan heuristic for the dcj median problem of genomes with unequal contents. *Computing and Combinatorics, LNCS*, 8591:227–238, 2014.

[258] Z. Yin, J. Tang, S.W. Schaeffer et D.A. Bader. Exemplar or matching : modeling dcj problems with unequal content genome data. *Journal of Combinatorial Optimization*, 2015.

[259] F. Yue, M. Zhang et J. Tang. Phylogenetic reconstruction from transpositions. *BMC Genomics*, 9(Suppl 2) :S15, 2008.

[260] R. Zeira et R. Shamir. Sorting by cuts, joins and whole chromosome duplications. *Combinatorial Pattern Matching, LNCS*, 9133:396–409, 2015.

[261] M. Zhang, W. Arndt et J. Tang. An exact solver for the dcj median problem. *In : Proc. 14th Pacific Symposium on Biocomputing PSB*, pages 138–149, 2009.

[262] D. Zhu et L. Wang. On the complexity of unsigned translocation distance. *Theoretical Computer Science*, 352:322–328, 2006.

[263] F.T. Zohora et M.S. Rahman. An efficient algorithm to detect common ancestor genes for non-overlapping inversion and applications. *Theoretical Computer Science. http ://dx.doi.org/10.1016/j.tcs.2016.03.006*, 2016.