Concepts et algorithmes pour la comparaison de séquences génétiques : une approche informationnelle

THÈSE

présentée et soutenue publiquement le 10 juillet 2000

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Jean-Stéphane VARRÉ

Composition du jury

Président :	Max DAUCHET, Professeur	LIFL, Université des Sciences et Technologies de Lille
Rapporteurs :	Maxime CROCHEMORE, Professeur	Institut Gaspard Monge, Université de Marne-la-Vallée
	Olivier GASCUEL, CR CNRS,	LIRMM, Université de Montpellier II
	Manolo GOUY, DR CNRS,	LBGP, Université Claude Bernard
Examinateur :	Didier BOUCHON, Professeur	LBPC, Univeristé de Poitiers
Directeurs :	Jean-Paul DELAHAYE, Professeur	LIFL, Université des Sciences et Technologies de Lille
	Eric RIVALS, CR CNRS	LIRMM, Université de Montepellier II

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille — UPRESA 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 20 43 47 24 – Télécopie : +33 (0)3 20 43 65 66 – email : direction@lifl.fr

Mis en page et composé avec LaTeX et la classe thlifl.

Table des matières

Introduction

Le chemin parcouru depuis la découverte de la structure de l'ADN par Watson et Crick, il y a un demi-siècle, est considérable. Alors que la recherche en génétique s'était orientée vers une analyse où, à partir d'une fonction connue on recherchait la séquence correspondante, l'abondance des données et leur traitement automatique permettent aujourd'hui une analyse où l'on recherche sur des séquences complètes les parties susceptibles de coder pour une fonction. Avec les projets de séquençage systématique (génome humain, programme Génoplante, etc) visant à obtenir la séquence complète de tous les chromosomes d'un organisme, la masse de données produite ne cesse de s'accroître, et ce de façon exponentielle. Organiser, analyser, structurer ces données implique la mise en place de traitements automatiques et offre un vaste champ de collaboration entre biologistes et informaticiens.

Le but de l'analyse à partir des séquences est de disposer de toute l'information génétique (au moins au niveau primaire) afin d'en faire une étude quasi-exhaustive. Il ne s'agit donc plus de séquencer spécifiquement une région du génome lorsqu'on en a besoin, quand on dispose de suffisamment d'informations, mais de rechercher à travers le génome complet où se situe l'information. Le séquençage massif fait donc que nous disposons d'une grande quantité de données sans connaître les informations que nous devons y associer.

Une approche simple pour obtenir de l'information à partir de séquences inconnues est de comparer ces séquences avec celles déjà connues, avec pour principe que si deux séquences se ressemblent, elles ont une forte probabilité de coder pour les mêmes fonctions. La comparaison de séquences représente ainsi un des domaines les plus actifs de la bioinformatique actuelle [Mye91, Wat95, SM97, Gus97]. Elle est le plus souvent envisagée au travers d'un alignement [NW70]. Cette vue permet de bonnes comparaisons dans le cas où les séquences sont proches, quand le nombre de modifications ponctuelles qui les séparent est réduit. Toutefois, elle ne permet pas la détection de similarités pour le cas de séquences plus éloignées. Des méthodes d'alignement local [SW81] peuvent alors être employées pour détecter les meilleurs segments similaires entre des séquences, au lieu de s'intéresser à celles-ci dans leur totalité. Les méthodes d'alignement présentent deux lacunes. D'une part, la comparaison est ponctuelle alors que l'identification de blocs similaires reflète mieux la réalité biologique. D'autre part, le mode de comparaison impose de respecter l'ordre de lecture des nucléotides dans les séquences, or certaines séquences évoluent en déplaçant ou en dupliquant des parties d'elles mêmes (réarrangements génomiques, transposons, etc). Différents travaux ont été menés pour palier à ces défauts. La méthode DIALIGN [MDW96] répond au premier d'entre-eux. Elle permet une comparaison des séquences par identification de blocs communs, mais elle est surtout dédiée à la production d'un alignement multiple. De ce fait, elle conserve l'ordre des bases et ne permet donc pas d'échanger des blocs. Les méthodes par réarrangements [Chr98] répondent au second. Elles permettent la comparaison de génomes grâce à des opérations modifiant l'ordre des gènes. Les gènes sont numérotés et les génomes sont

donnés sous la forme d'une suite de gènes. Leur défaut est que toute l'information relative à la constitution des gènes et aux parties inter-géniques n'est pas prise en compte.

Le projet de la thèse est la définition de méthodes permettant la comparaison de séquences au niveau de la structure primaire, grâce à la détection de segments communs, et en autorisant les segments communs à ne pas être dans le même ordre dans les séquences. Pour cela, nous avons abordé le problème sous un angle radicalement différent des méthodes employées généralement pour la comparaison de séquences.

La théorie de la **complexité de Kolmogorov** [LV93] mesure la quantité d'information contenue dans un objet par la longueur de sa plus courte description. La complexité relative de Kolmogorov est définie de manière similaire comme la longueur de la plus courte description d'un objet relativement à un autre, c'est-à-dire en supposant que cet autre objet est connu. Elle permet ainsi la mesure de la quantité d'information d'un objet relativement à un autre, c'est-à-dire la mesure de la quantité d'information qui doit être ajoutée à la description du premier objet pour obtenir la description du second.

Bien que la complexité de Kolmogorov ne soit pas calculable, l'élaboration d'**algorithmes de compression** adaptés aux problèmes que l'on traite permet d'en obtenir une majoration. En effet, un algorithme de compression est un processus permettant de réécrire un objet (un texte, une image, etc) en exploitant sa structure (répétitions, duplications, etc). Lorsqu'on tente de comprimer un objet, on cherche donc à obtenir une description la plus courte possible de cet objet. Malheureusement, un algorithme de compression suit toujours un **schéma de compression**, c'est-à-dire un ensemble de structures exploitées, par exemple des duplications, bien défini par avance, et dont on ne peux pas prouver qu'il décrive un objet de la manière la plus compacte qui soit. La description d'un objet comprimé n'est donc qu'une approximation de la description associée à la mesure de complexité de Kolmogorov.

Dans ce cadre de compression de données, nous avons défini une mesure de dissimilarité, la **distance de transformation**[1], qui mesure la quantité minimale d'information qui doit être ajoutée à une séquence source S pour obtenir une séquence cible T. La valeur de la distance est obtenue grâce à la mesure de la longueur d'un **script de transformation**, qui est une suite d'opérations dont l'application à la séquence S permet d'obtenir la séquence T.

Parallèlement à la notion de schéma de compression, nous avons défini la notion de **système générateur** qui représente un ensemble d'opérations qu'il est possible d'utiliser dans le script. Ainsi, c'est une famille de distances qui est définie, chaque système générateur spécifiant une distance particulière. Nous nous sommes plus particulièrement attachés à l'étude de la **distance par copies de facteurs**, qui est une spécification définie sur un système générateur comportant deux opérations : l'une recopiant les parties communes à S et T, l'autre créant les parties propres à T.

Les distances définies fournissent bien une mesure de la dissimilarité entre les séquences puisqu'elles mesurent la partie d'information de T qui n'est pas présente dans S. En plus de cette mesure, le script, en tant que liste de segments communs, fournit une évaluation qualitative de la ressemblance entre les deux séquences. Nous disposons ainsi d'une mesure basée sur des critères objectifs de partage d'information, s'affranchissant des limitations de respect de l'ordre des bases

1. Le terme de distance doit être pris au sens large et pas réduit à la distance mathématique.

dans les séquences, s'appuyant sur l'identification d'une ressemblance par blocs et proposant un schéma de similitude associé à l'évaluation quantitative de la ressemblance.

Nous avons appliqué la distance de transformation à l'étude de la phylogénie des isopodes terrestres [MSB00]. Les résultats sont très encourageants puisque la majorité des relations phylogénétiques que nous proposons sont congruentes avec les connaissances biologiques et les arbres reconstruits à partir des méthodes traditionnelles. Cela met en évidence la capacité de notre mesure à évaluer la ressemblance entre deux séquences. Grâce à la distance de transformation, il est ainsi possible de calculer pour un ensemble de séquences, sans alignement préalable et à partir des séquences prises dans leur totalité, une matrice de distances proposant une mesure effective des ressemblances entre les séquences.

Plan de lecture

Nous débutons ce document par une brève description des objets que nous allons manipuler : les séquences génétiques. Celle-ci donne quelques rappels de notions fondamentales en biologie et décrit rapidement les différents modes d'évolution des séquences : les mutations ponctuelles et les réarrangements de segments. Cela permettra de comprendre l'intérêt et les difficultés de la comparaison de séquences.

Comparaison de séquences et point de vue informationnel

Le chapitre 2 s'intéresse à la comparaison des séquences génétiques. Nous discutons des modèles existants pour la comparaison de ces objets biologiques. En particulier, nous expliquons le principe des méthodes d'alignement qui définissent les **distances ponctuelles** : aligner deux séquences c'est trouver une mise en correspondance des lettres des deux séquences en s'autorisant des insertions de blancs dans les deux séquences (les insertions et les délétions) et en permettant à deux lettres différentes d'être en relation (les substitutions). On définit un score d'alignement en attribuant un poids à chaque opération (insertion, délétion et substitution), donc à chaque couple de lettres, et en sommant les poids de toutes les opérations pour tous les couples de lettres. Nous décrivons les deux algorithmes principaux permettant le calcul de ce score : celui de Needleman et Wunsh [NW70] pour la comparaison globale et celui de Smith et Waterman [SW81] pour la comparaison locale. Nous abordons ensuite les notions de **distances par réarrangements** (distances par inversions, translocations, etc) [Chr98, KS93, Han95a] développées lorsqu'il s'agit de mesurer la ressemblance entre génomes grâce à des opérations agissant sur des segments. Il s'agit cette fois de trouver une transformation minimale permettant de passer d'un génome, vu comme une suite d'objets (le plus souvent des gènes), à un autre. Cela revient donc à trouver un **réarrangement** d'une suite d'objets dans un ordre donné en la même suite dans un autre ordre grâce à l'application d'une ou plusieurs opérations. Nous terminons par un aperçu des distances utilisées pour la phylogénie [SOWH96, DT93, Ber98]. Celles-ci sont basées sur des distances ponctuelles et proposent des corrections du score d'alignement en supposant certains **modèles d'évolution**.

Le chapitre 3 aborde la mesure de la ressemblance d'objets du point de vue mathématique. Nous discutons principalement du concept de la **distance informationnelle** [BGL+98] dont le fondement est que la mesure de la ressemblance entre objets est liée à la quantité d'information que partagent ces objets. Celle-ci est définie grâce à la complexité de Kolmogorov et plus particulièrement à la **complexité de Kolmogorov relative** qui fournit une mesure de la quantité

d'information qu'il faut ajouter à un objet donné pour en obtenir un autre. En comprimant un objet cible en utilisant des informations d'un objet source, alors nous exploitons les ressemblances qui existent entre ces deux objets. La mesure de la quantité d'information qu'il faut ajouter à cet objet source fournit une majoration de la complexité relative de Kolmogorov entre ces deux objets. Nous expliquons les concepts de la compression de données dans le cas où elle sert à la compréhension des objets et non à réduire l'espace qu'occupent ceux-ci. Nous terminons par quelques exemples de travaux où les concepts de théorie de l'information et de compression de données sont utilisés pour l'analyse d'objets biologiques [RDDD96, MJ93, Tah97, LHYN95, Del97, APD99].

La distance de transformation

Après ce tour d'horizon des méthodes de comparaison, nous exprimons dans le chapitre 4 l'idée de **distance de transformation** que nous avons définie dans le cadre de cette thèse pour la comparaison de séquences génétiques. Nous introduisons la notion de **script de transformation**, permettant grâce à un ensemble d'opérations nommé **système générateur**, de transformer une séquence S en une séquence T. Nous définissons également la notion de **script de reconstruction** qui est un cas particulier de transformation où l'obtention de T se fait par ajouts successifs de segments. En associant un poids approprié aux opérations, il est possible d'obtenir une mesure, égale à la somme des poids des opérations, qui permet de quantifier la ressemblance entre S et T. La distance est définie comme le poids du script de poids minimal donnant T à partir de S, le script associé est appelé **script minimal**. Les fonctions de poids sont définies dans un cadre informationnel, et le poids du script est une mesure de la quantité minimale d'information qu'il faut ajouter à la donnée de S pour obtenir T, pour un système générateur et un jeu de poids donnés. Après avoir exposé ces notions, nous détaillons les relations très étroites qui existent entre la théorie algorithmique de l'information et la compression de données. Nous montrons ainsi comment la distance que nous définissons est effectivement une mesure de la similitude entre deux séquences. La distance de transformation définit une famille de distances. Une distance particulière est obtenue en spécifiant un système générateur et des fonctions de poids associées. Nous nous sommes intéressés au cas particulier de recherche d'un script de reconstruction où seulement deux opérations sont permises, l'une recopiant des parties communes à S et T (opération "copie") et l'autre permettant la création des autres parties de T (opération "insertion"). La définition des poids est abordée sommairement dans ce chapitre et décrite en détails au chapitre 6. C'est à partir de cette spécification "minimale" que nous avons mis en œuvre l'implantation du calcul de la distance.

Nous présentons ensuite un algorithme capable de calculer la distance par copies de facteurs à partir d'une représentation des scripts dans un graphe orienté, pondéré, acyclique et à source unique : le **graphe de scripts**. Chaque nœud représente une opération possible de copie tandis que chaque arc représente l'insertion nécessaire entre les deux copies représentées par les sommets source et destination de l'arc. Un chemin dans ce graphe représente ainsi un script possible permettant de construire T à partir de la donnée de S. En associant à chaque nœud le poids de la copie correspondante et à chaque arc le poids de l'insertion correspondante, le poids d'un chemin, égal à la somme des poids des nœuds et des arcs, correspond au poids du script associé. Le graphe de scripts contenant tous les scripts possibles, une recherche du chemin le plus court, qui correspond au script de poids le plus faible, permet de trouver le script minimal. Des algorithmes classiques de recherche de plus court chemin [CLR94] permettent de calculer la distance en temps linéaire en fonction du nombre d'arcs et du nombre de sommets, et donc polynomial

en fonction de la longueur des séquences. Malheureusement, ce nombre est très grand si l'on effectue une recherche exhaustive de tous les scripts possibles à partir de la donnée de toutes les parties communes, que nous appelons **facteurs**, entre S et T. Nous proposons donc deux algorithmes plus rapides en pratique. Le premier est basé sur la mise en avant de propriétés de scripts non optimaux, il aboutit à la définition de **graphe de scripts compact**. Le second se base sur un ensemble donné de facteurs, par exemple des facteurs biologiquement pertinents. On suppose que tout facteur qui peut être inclus dans un script dois l'être. Il aboutit à la définition de **graphe de scripts valides**. L'exactitude de ces algorithmes dépend de conditions sur les fonctions de poids associées aux opérations pour certaines topologies de facteurs. Nous énonçons celles-ci et démontrons des propriétés permettant de supprimer certains nœuds et certains arcs dont les opérations de copie et d'insertion associées ne peuvent appartenir au script minimal. Si ces conditions sont vérifiées, la distance calculée correspond bien au poids du script minimal. Dans le cas contraire, les algorithmes sont heuristiques.

La définition de la distance nécessite d'être spécifiée un peu plus pour obtenir une véritable mesure. Dans le chapitre 6, nous discutons des choix faits pour la mesure implémentée. Nous discutons d'abord du **choix des facteurs** qui n'est pas trivial, puisqu'il conditionne directement les similitudes qui seront mises en avant. Conjointement à cela, nous abordons le problème d'une borne minimale pour la **longueur des facteurs**. Il faut que cette borne soit satisfaisante d'un point de vue conceptuel, c'est-à-dire qu'elle ne doit ni supprimer d'éventuelles vraies similitudes, ni introduire de faux facteurs, dont la ressemblance provient du hasard. Mais cette borne doit également satisfaire des contraintes liées à la praticabilité du calcul. Nous montrons que des copies de facteurs de taille supérieure ou égale à trois permettent d'obtenir une valeur de distance plus petite car le choix d'un facteur dépend de l'environnement dans lequel il se trouve, c'est-à-dire, s'il est entouré par d'autres facteurs. Nous introduisons la notion de **point d'effort** définie comme la valeur de longueur des facteurs en deçà de laquelle le gain sur la valeur de la distance est faible comparé à la perte en temps d'exécution. Nous corrélons cette valeur avec la borne théorique de longueur moyenne de facteurs dans deux séquences aléatoires. Nous abordons ensuite le problème du **choix des fonctions de poids**. Nous montrons que celui-ci influence grandement la notion de similitude captée par la distance. Nous proposons une évolution d'une fonction de poids basique vers une fonction qui paraît assez satisfaisante, mettant en avant des propriétés de ressemblance qui nous semblent justifiées : avantage aux copies de facteurs avec écarts quasi-constant, croisement un peu plus cher, désavantage à la copie d'un facteur dont les segments sont à des positions éloignées, etc. Nous poursuivons par une étude des propriétés de la distance. En particulier, nous expliquons les raisons intrinsèques de son asymétrie. Nous évaluons les différences dues aux fonctions de poids par quelques analyses expérimentales. Nous terminons en étudiant la qualité des résolutions par les différentes méthodes de graphes de scripts. Nous nous intéressons d'abord à la vérification des conditions liées aux propriétés à partir desquelles ont été définis les deux algorithmes du graphe de scripts compact et du graphe de scripts valides. Certaines d'entre-elles ne sont malheureusement pas vérifiées et les algorithmes sont donc heuristiques. Nous nous intéressons alors à la comparaison des temps de calcul et des taux d'erreurs liés à l'utilisation de ces deux algorithmes heuristiques. Nous montrons qu'avec le graphe de scripts compact, les gains en temps sont considérables pour une erreur négligeable. Le calcul de toutes les comparaisons pour un ensemble de 46 séquences de longueur 400pb nécessite seulement 4 minutes de calcul, contre une heure pour le graphe de scripts, avec uniquement 2,7% des cas avec erreur et un taux d'erreur atteignant au maximum 2%.

Une application à la reconstruction phylogénétique des isopodes terrestres

Le dernier chapitre est consacré à une étude expérimentale d'un jeu de données biologiques. Il s'agit de 46 séquences d'ARNr 16S mt de crustacés (la grande sous-unité de l'ARN ribosomique mitochondrial). La position des isopodes terrestres au sein des isopodes est controversée aujourd'hui et les études morphologiques n'ont pas permis d'aboutir à un consensus. Alice Michel-Salzat et Didier Bouchon du Laboratoire de Génétique et Biologie des Populations de Crustacés, UMR CNRS 6556, de l'Université de Poitiers, ont entrepris la reconstruction d'une première phylogénie à partir de données moléculaires [MSB00]. Leur étude tente de répondre à des questions concernant la répartition des *Oniscidea*, sous-ordre des isopodes contenant en majorité des espèces d'isopodes terrestres, au sein des isopodes. Nous avons calculé la matrice de distance entre tous les couples de séquences avec la distance de transformation (à partir des séquences brutes, sans alignement préalable), puis reconstruit une phylogénie à l'aide de la méthode Neighbor-Joining [SN87]. Les études que nous avons menées au chapitre 6, et des expériences menées sur les données, ont permis de fixer les paramètres de notre méthode (fonction de poids, longueur minimale des facteurs communs, etc). Nous avons ensuite analysé plusieurs reconstructions avec des échantillonnages d'espèces différents, afin de mesurer la qualité de la phylogénie produite. Les relations phylogénétiques que nous proposons confirment les connaissances acquises jusqu'à maintenant sur ces espèces, ce qui est une validation de la capacité de notre méthode à mesurer correctement la ressemblance.

La distance de transformation permet donc de mesurer la ressemblance entre deux séquences, même si celles-ci ne sont pas "alignables". Le script minimal donne une description sous la forme d'un schéma de recombinaison de facteurs communs identifiés comme les plus valables. La mesure associée permet de quantifier le degré de ressemblance des séquences étudiées.

La capacité de la distance de transformation à évaluer la ressemblance entre deux séquences permet de l'utiliser dans différents types d'applications. Elle peut être ainsi utilisée pour la comparaison de séquences. Elle permet alors de repérer des parties communes à deux séquences qui n'auraient pas été identifiées par un alignement à cause de la contrainte du respect de l'ordre des bases, par exemple des duplications. En s'affranchissant de tout travail préliminaire sur les séquences, son utilisation en fait un bon outil d'appréciation objective d'un schéma de ressemblance. Sa modularité fait qu'elle peut être utilisée pour l'étude de petites séquences, tels que des ARN ribosomiques, ou pour l'étude de séquences plus grandes, telles que des gènes ou des chromosomes. Comme l'a montré l'application faite au chapitre 7, elle peut également être utilisée pour la reconstruction d'arbres phylogénétiques.

De nombreuses spécifications du concept de distance de transformation sont envisageables. La définition de systèmes générateurs et de fonctions de poids dédiés à l'étude d'un problème particulier permettent d'ajuster la méthode au contexte dans lequel elle doit s'appliquer. Ainsi, on n'adoptera pas la même approche pour mesurer la ressemblance lorsqu'il s'agira de comparer des séquences courtes, telles que des gènes, que des séquences longues, telles que des chromosomes ou des génomes complets.

Chapitre 1

Notions de biologie

Nous décrivons brièvement dans ce chapitre quelques notions de biologie à l'usage des informaticiens. Le but est de donner juste assez d'éléments de biologie moléculaire pour permettre la compréhension des problèmes et des enjeux de cette thèse. C'est pourquoi nous n'insistons pas sur les détails et que la vue que nous proposons est parfois schématique, voire simpliste.

1.1 Les molécules de la vie

De manière très simplifiée, la vie des organismes, qu'ils soient primitifs ou complexes s'articule autour de deux principales entités biochimiques : les **acides aminés** et les **acides nucléiques**.

Ces molécules s'assemblent pour former des chaînes : d'une part, les **protéines** (chaînes d'acides aminés) qui sont les acteurs du métabolisme, et d'autre part l'**ADN** et l'**ARN** (chaînes de nucléotides). L'ADN est le support de l'information génétique héréditaire et l'ARN transporte le message jusqu'au point de construction des protéines et relie le génotype au phénotype.

Les protéines

Une protéine est une molécule formée d'une chaîne d'**acides aminés**, qui code pour des **fonctions**. Il existe vingt acides aminés, à chacun est associé une lettre permettant l'écriture d'une protéine sous forme d'un texte linéaire. On notera $\mathcal{A}$ l'alphabet des acides aminés. Une protéine est un mot écrit sur cet alphabet. La longueur d'une chaîne protéique varie entre 100 et 5000 acides aminés et une longueur moyenne est de 300 acides aminés.

Mais une chaîne protéique n'est pas la simple donnée d'une suite d'acides aminés. Les conformations imposées par la biochimie font que l'on distingue plusieurs "niveaux" de la protéine, appelées **structures**. La séquence d'acides aminés, c'est-à-dire la suite des lettres représentant les acides aminés, est la **structure primaire**. Il existe trois autres niveaux définissant la structure secondaire, tertiaire et quaternaire.

En particulier, on pourra noter que la structure tridimensionnelle (tertiaire) d'une protéine joue un rôle important dans la fonction codée par celle-ci. C'est d'ailleurs l'un des domaines de recherche très actif de la bioinformatique.

Un acide aminé est le résultat du "décodage" d'un triplet d'acides nucléiques. Un tel triplet

est appelé **codon** (voir tableau 1.1). Ce décodage est réalisé durant la phase de traduction par des ARN de transfert (voir la section 1.3).

Première position	Seconde position				Troisième position
	G	A	C	U	
G	Gly	Glu	Ala	Val	G
					A
		Asp			C
					U
A	Arg	Lys	Thr	Met	G
				Ile	A
	Ser	Asn			C
					U
C	Arg	Gln	Pro	Leu	G
					A
		His			C
					U
U	Trp	STOP	Ser	Leu	G
	STOP				A
	Cys	Tyr		Phe	C
					U

TAB. 1.1 – *Le code génétique : les acides aminés et leurs codons.* `CCG` *code par exemple pour la proline (Pro) et* `AUG` *pour la methionine (Met).*

L'ADN (acide desoxyribonucléique)

Tout comme les protéines, l'ADN est une chaîne de molécules plus simples : les **acides nucléiques**. La chaîne d'ADN est le support de l'information génétique. Cette information est traduite au travers de mécanismes que nous évoquerons plus loin. Les acides nucléiques sont souvent confondus avec les **bases** qui en sont des constituants (voir figure 1.1). Il existe quatre bases : Adénine (A), Cytosine (C), Thymine (T) et Guanine (G). On notera $\mathcal{N}$ l'alphabet des acides nucléiques. La longueur des chaînes d'ADN est très variable : de quelques dizaines de bases (on parlera d'**oligonucléotides**) à plusieurs millions. L'unité de mesure est la paire de bases (voir ci-dessous), notée pb.

L'ADN possède la particularité de former une double hélice (voir figure 1.3). C'est-à-dire qu'une molécule d'ADN n'est pas une simple chaîne comme le sont les protéines mais une double chaîne ; chaque chaîne est appelée **brin**. Ces deux chaînes sont reliées par un mécanisme d'appariement découvert par Watson et Crick en 1953. Ainsi une base A est appariée avec une base T et une base C avec une base G , et inversement (voir figure 1.2). On dit alors que les bases A et T, respectivement C et G, sont **complémentaires**.

Lorsque l'on parle du **complémentaire** d'un segment d'ADN, on fait référence au segment obtenu en prenant le complémentaire de chaque base et en renversant l'ordre des bases. Si u =`ATGGTGATGCCT`, le complémentaire sera : `AGGCATCACCAT`.

Enfin les bases sont classées par famille : A et G sont les **purines** et C et T sont les **pyrimi-**

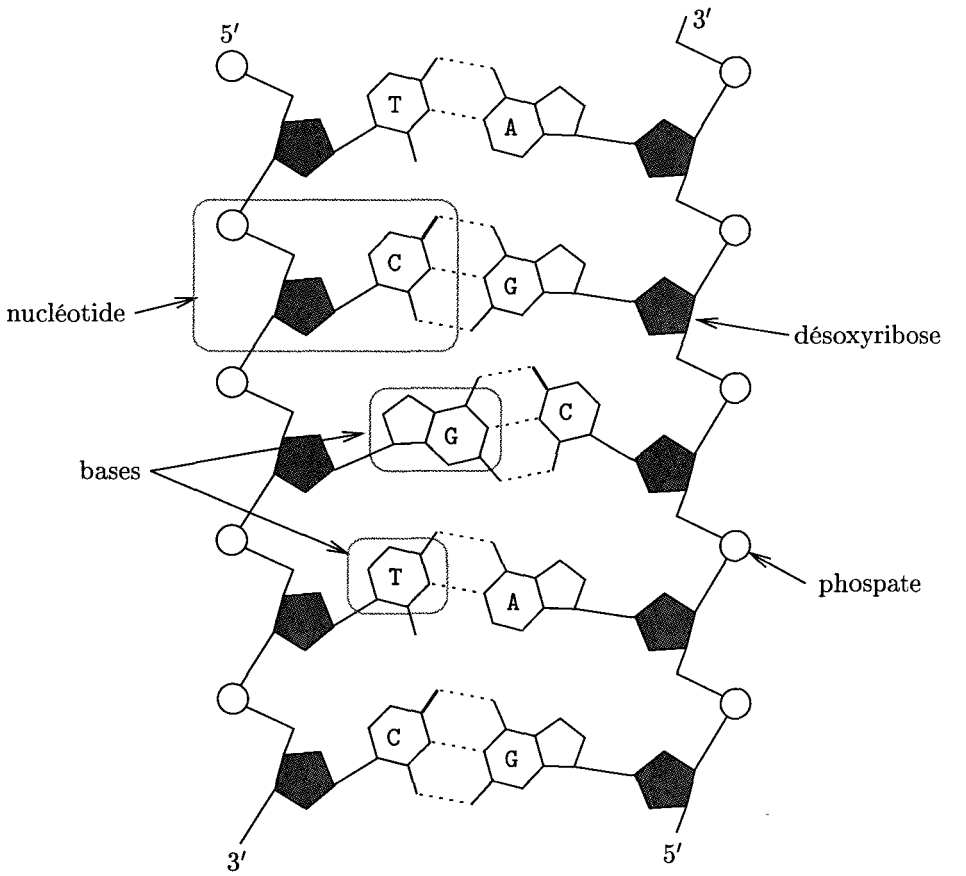


FIG. 1.1 – *Représentation de la structure de la double hélice d'ADN.*

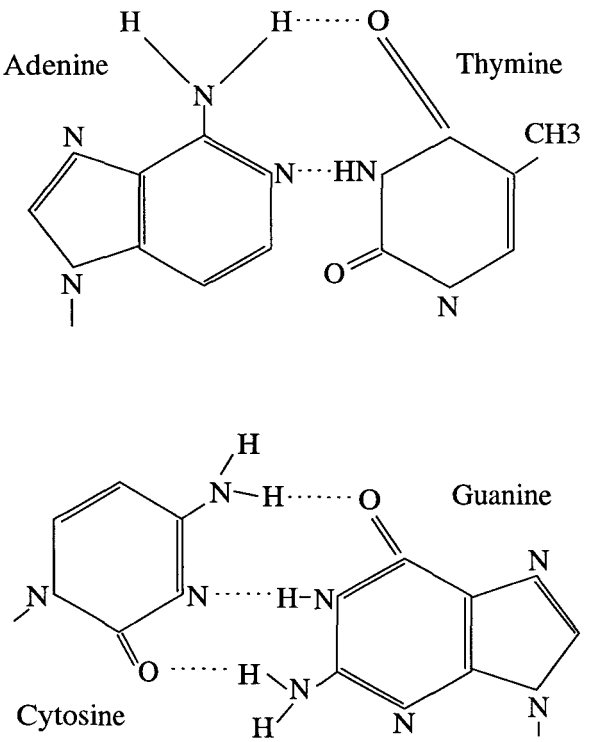


FIG. 1.2 – *Les acides nucléiques. La liaison entre les acides aminés est possible grâce aux liaisons dessinées en pointillés. La liaison C-G est plus forte que la liaison A-T.*

dines. Il est également possible que les bases s'associent dans leur famille, ce sont les transitions : A avec G ou C avec T. Les deux transversions A avec C et G avec T existent également. Cela est dû à des modifications pontuelles de la séquence des acides nucléiques (voir section 1.4).

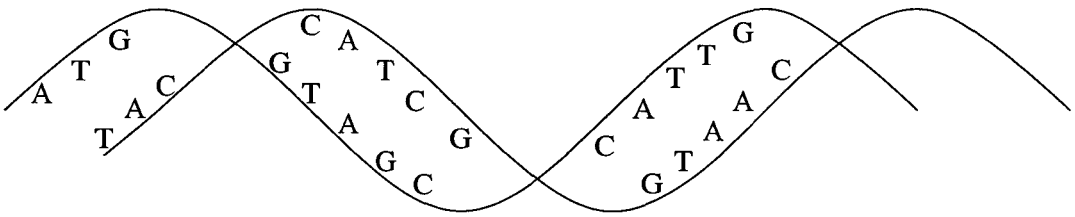


FIG. 1.3 – *La double hélice d'ADN.*

La synthèse de molécules d'ADN filles s'effectue grâce au processus de **réplication**. Les deux brins d'une molécule d'ADN sont séparés puis deux nouvelles molécules sont obtenues par reconstitution "en face" de chaque brin du brin complémentaire (voir figure 1.4).

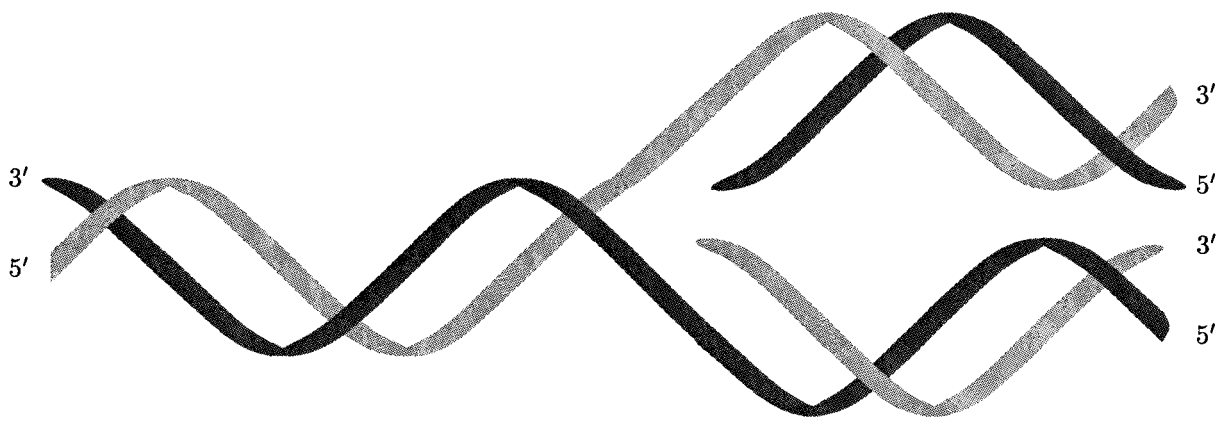


FIG. 1.4 – *Réplication d'une molécule d'ADN.*

La conformation biochimique des acides nucléiques fait qu'il y a une **orientation** de chacun des brins d'ADN. On différencie une extrémité noté 5' de l'autre notée 3'. Par convention les molécules d'ADN sont toujours représentées par un seul de leur brin (l'autre étant obtenu par complémentarité), celui possédant l'orientation $5' \longrightarrow 3'$.

L'ARN (acide ribonucléique)

L'ARN ressemble fort à l'ADN, il est également composé de quatre bases mais l'Uracile (U) remplace la Thymine (T), et est constitué d'une seule chaîne.

1.2 Les vecteurs du patrimoine génétique

chaîne L'ADN est le principal support de l'information génétique puisque l'ARN, puis les protéines résultent de sa transformation comme nous le verrons dans la section suivante. Le

maintien, le support et la transmission de cette information à une descendance se fait au travers des **chromosomes** et des **gènes**.

Chaque cellule d'un organisme vivant contient quelques molécules d'ADN. Ce nombre varie suivant l'espèce, pour l'homme il est de 46. Chacune de ces molécules est appelé un **chromosome**. L'ensemble des chromosomes forme le **génome**. Certains segments de l'ADN des chromosomes contiennent l'information nécessaire à la production des protéines, ce sont les **gènes** alors que d'autres segments n'en contiennent pas. Typiquement, chaque protéine d'un organisme correspond à un gène.

Chez les organismes "évolués", les **eucaryotes**, les gènes sont eux-mêmes organisés en différentes parties : les **introns** sont les parties qui ne sont pas traduites et les **exons** sont celles qui le sont. Les exons sont donc les seules parties de l'ADN qui servent à coder pour les protéines. Chez les organismes non eucaryotes, les **procaryotes**, les gènes sont entièrement codants.

1.3 De l'ADN à la protéine

Nous pouvons maintenant décrire brièvement les mécanismes qui permettent d'extraire l'information de l'ADN pour produire les protéines.

Le premier mécanisme de transformation de l'ADN est le processus de **transcription**. Ce processus permet de créer une molécule d'ARN à partir de l'ADN. Cet ARN est appelé **ARN messager** et noté ARNm. Il correspond au complémentaire de l'un des deux brins d'ADN, en remplaçant T par U. Chez les eucaryotes, c'est à ce moment que les introns sont éliminés, l'ARNm n'est que le complémentaire de tous les exons mis bout à bout : c'est la phase d'**épissage**.

Une fois cet ARN messager produit, il va servir de "plan de construction" pour la production des protéines au cours de la **traduction**. Cette transformation s'effectue au sein des **ribosomes** (voir figure 1.5). Cette opération nécessite également une "matière première" : l'**ARN de transfert** noté ARNt. L'ARNt décode l'association entre un codon et son acide aminé, c'est-à-dire le **code génétique** (voir table 1.1).

Le fonctionnement du ribosome peut maintenant être décrit simplement. L'ARN messager passe au travers du ribosome. A chaque fois qu'un codon le traverse, un ARN de transfert le reconnaît et l'acide aminé correspondant est ajouté à la protéine en cours de construction. Si un codon STOP est lu, la synthèse de la protéine s'arrête.

La figure 1.6 résume le processus de passage de l'ADN à la protéine.

1.4 Evolution des séquences

Il existe de nombreuses sources de mutations permettant l'évolution des séquences. Ces mutations peuvent se produire lors de la réplication ou lors des phases de réparation de l'ADN. Elles sont de deux types : soit ponctuelles, elles ne concernent qu'un résidu de la molécule; soit elles concernent plusieurs résidus adjacents (un segment de la molécule), nous les dénomerons mutations multinucléotides.

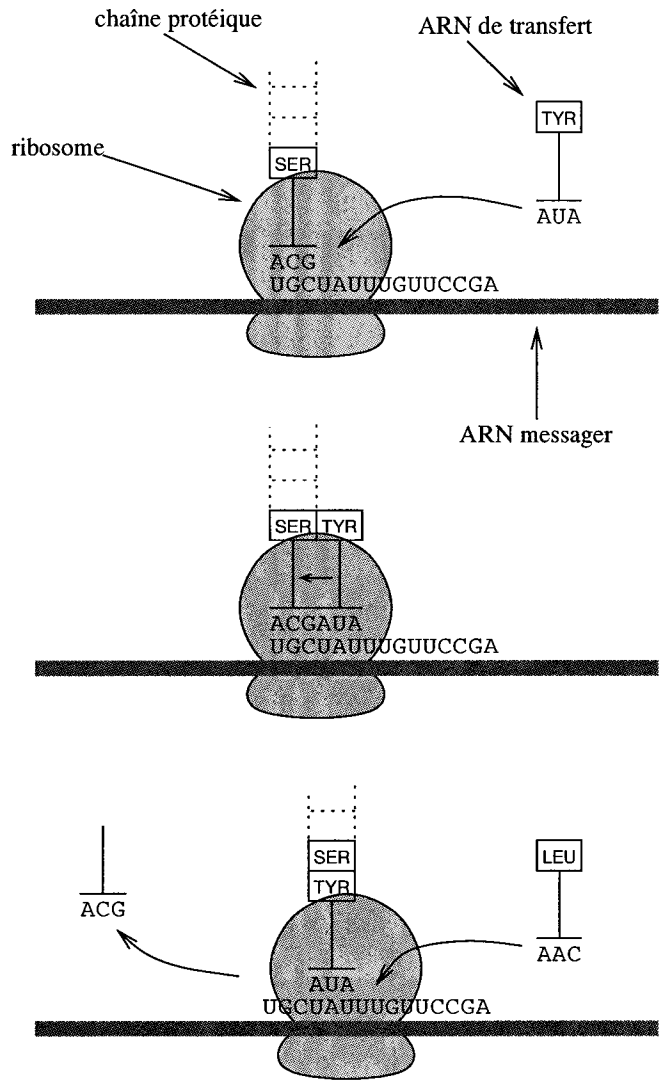


FIG. 1.5 – *Traduction de l'ARN messager en protéine. Le ribosome se déplace le long de la molécule d'ARN messager. A chaque codon est associé un ARN de transfert. Cet ARNt possède un anticodon capable de s'attacher au codon de l'ARNm. Une liaison est alors construite entre la protéine portée par l'ARNt et la chaîne protéique déjà construite.*

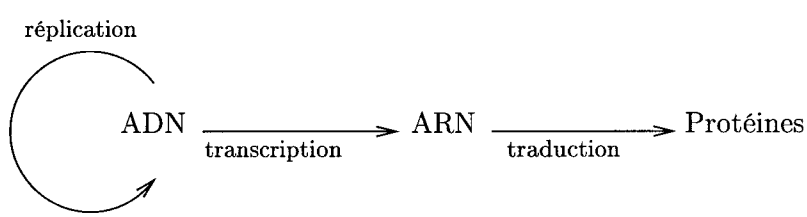


FIG. 1.6 – *Dogme central : vue schématique du processus de transformation de l'ADN en protéines.*

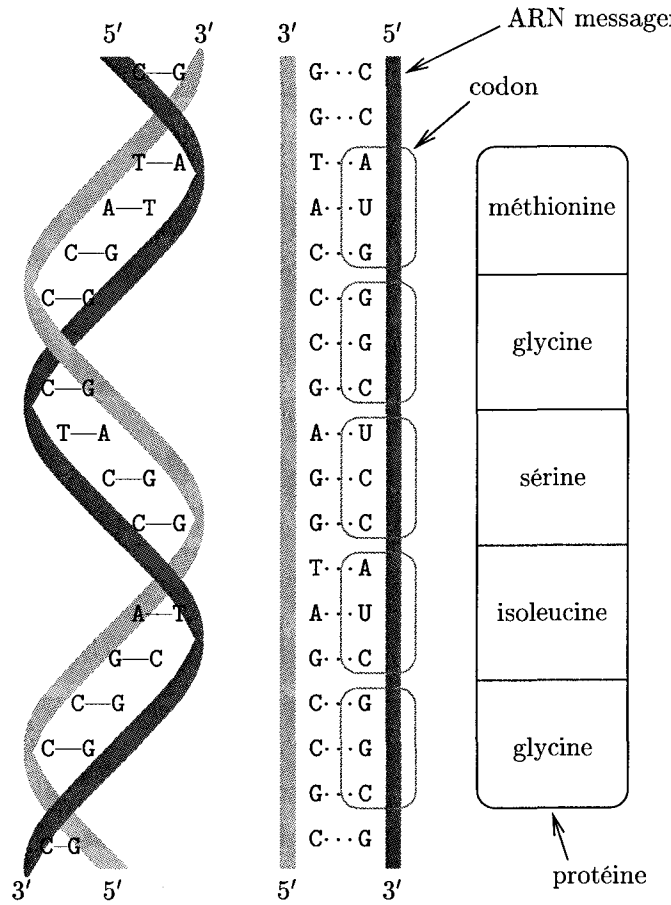


FIG. 1.7 – *Schéma de la transcription du message génétique. L'ADN est d'abord traduit en ARN messager, qui est le complémentaire du brin appelé matrice. Le simple brin d'ARN se détache et est ensuite traduit en protéine : chaque codon est traduit en l'acide aminé lui correpondant selon le code génétique.*

Mutations ponctuelles. Elles sont la modification d'une base de l'ADN. Cette modification peut être une substitution (modification d'une base en une autre), une insertion ou une délétion. S'il s'agit d'une insertion ou d'une délétion, le code génétique est bien sûr modifié puisque, comme nous l'avons vu, les acides aminés sont codés par des triplets d'acides nucléiques. Cependant, si la mutation est une substitution, il se peut que la modification affecte ou n'affecte pas l'acide aminé codé par le triplet. Par exemple une mutation du `C` dans le codon `GTC` le modifie en `GTA` qui code aussi pour la Valine (voir table 1.1). Si elle ne l'affecte pas, la mutation est dite **synonyme** ou **silencieuse**. Dans les autres cas, la mutation sera dite non-synonyme et peut être subdivisée en deux types: celles d'erreur de sens font que l'acide aminé codé par le triplet va être modifié (par exemple `GTC` code pour Valine et `TTC` code pour Phenylalanine); celles de non-sens font que le triplet va coder pour un codon STOP (par exemple `AAG` code pour Lysine et `TAG` est un codon STOP). La mutation à l'intérieur d'une famille (purine ou pyrimidine) est une **transition** (`A` en `T` et inversement ou `C` en `G` et inversement) alors qu'une mutation entre des familles est une **transversion** (`A` ou `T` en `G` ou `C`, et inversement). Il faut noter que la position de la mutation dans le codon est importante. En effet, si l'on regarde la table 1.1, on s'aperçoit que les mutations en troisième position ont une plus faible probabilité d'affecter le code génétique que celles en première position.

Ces mutations interviennent principalement lors de la réplication de l'ADN. Ce type de mutation est étudié et détecté par les méthodes d'alignement (voir section 2.1.4).

Ces modifications de l'ADN se situent au niveau le plus bas, c'est-à-dire au niveau du résidu. Il existe d'autres types de mutations qui cette fois manipulent des parties entières du génome.

Remaniements chromosomiques. Les mutations dont nous parlons maintenant concernent donc des segments entiers d'ADN. On appelle **remaniement** toute modification d'un chromosome qui modifie la séquence linéaire de ses gènes. On distingue plusieurs types de remaniements:

- les **inversions** permettent de retourner un segment d'un chromosome. Le segment résultant de l'inversion est le complémentaire inversé du segment initial,
- les **translocations** correspondent à un échange entre les deux extrémités de deux chromosomes,
- les **transpositions** sont l'échange de deux parties adjacentes d'un même chromosome,
- les **délétions** et les **insertions** sont respectivement la suppression ou l'ajout d'un segment à une position donnée d'une séquence.

La figure 1.8 représente quelques unes de ces mutations.

Eléments transposables Leur découverte a été faite par Barbara McClintock à la fin des années 40. Ces éléments ont la capacité de se déplacer ou de se multiplier dans un génome hôte. Ces éléments sont très nombreux chez les eucaryotes, ils représentent par exemple 10 à 12% du génome de la drosophile. Ils sont par conséquent considérés aujourd'hui comme des acteurs majeurs de l'évolution.

Ces éléments sont classés en deux catégories suivant qu'ils se propagent par un intermédiaire ADN ou un intermédiaire ARN. La figure 1.9 résume le mode de propagation des différents éléments.

Par ailleurs les éléments transposables peuvent également être la cause de remaniements chromosomiques lors de la recombinaison (voir figure 1.10).

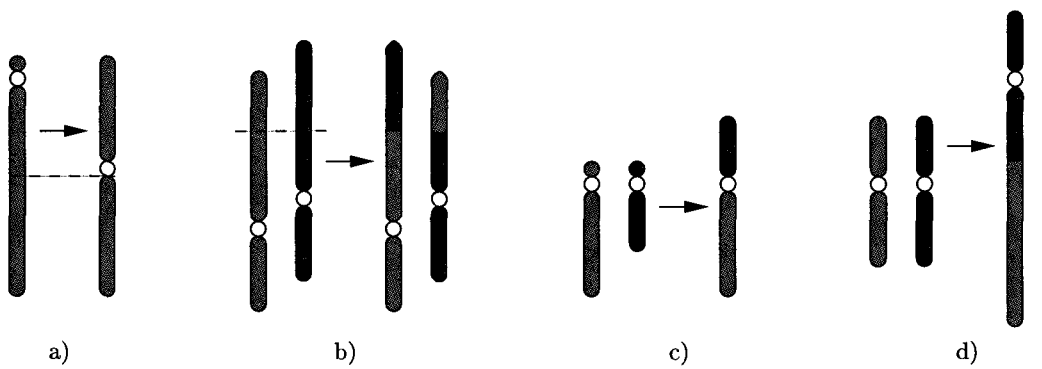


FIG. 1.8 – *Remaniements chromosomiques : a) inversion, b) translocation réciproque, c) translocation robertsonienne, d) translocation en tandem.*

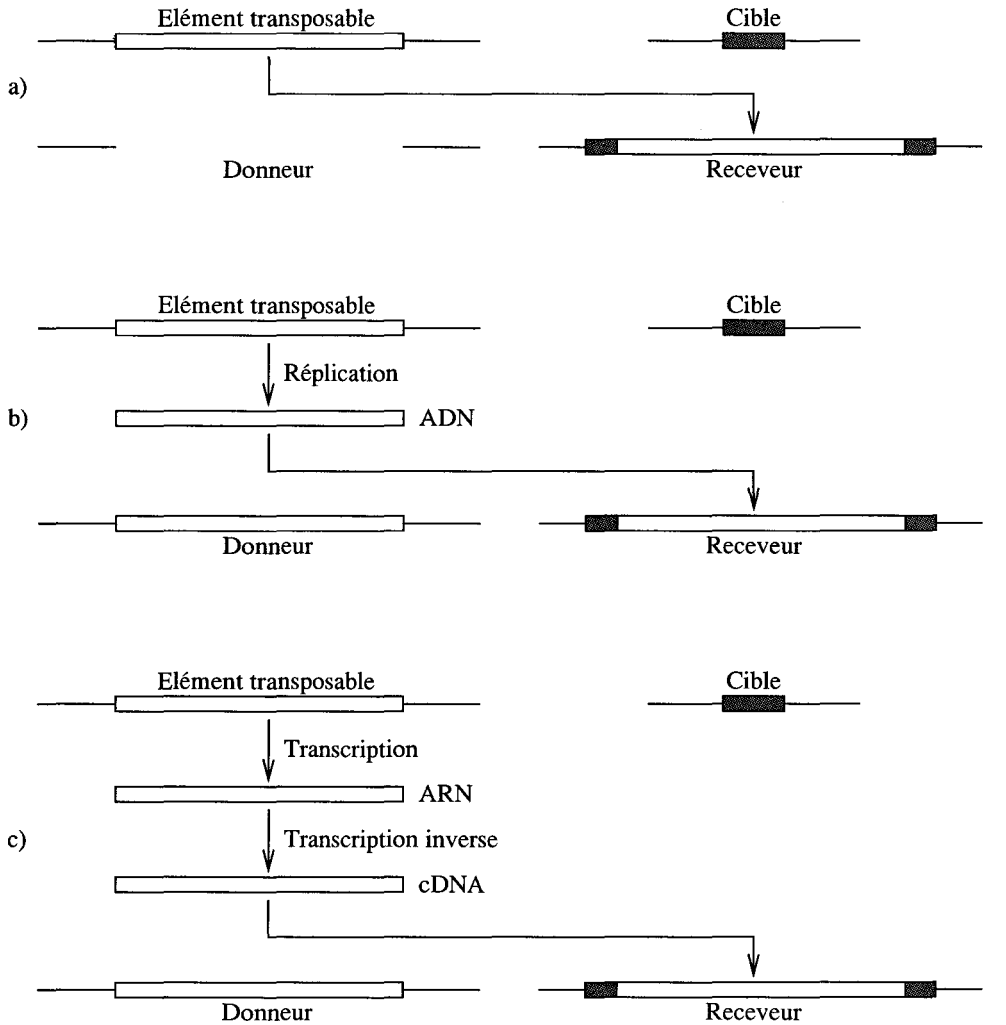


FIG. 1.9 – *Modes de propagations des éléments transposables (tiré de [Li97]). a) transposition non réplicative : l'élément transposable est déplacé à un autre enplacement, on ne sait pas ce que devient la séquence donatrice, b) transposition réplicative : l'élément transposable est répliqué puis inséré, c) rétroposition : l'élément transposable est d'abord transcrit en ARN puis la trancription inverse en une copie ADN qui est insérée.*

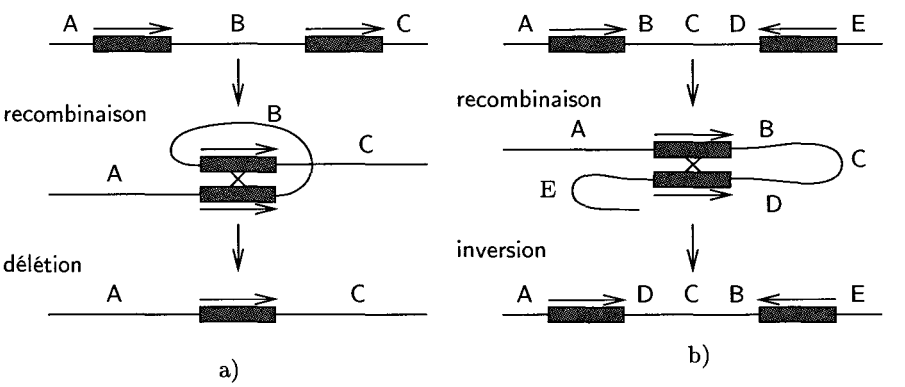


FIG. 1.10 – *Remaniements chromosomiques causés par les transposons. a) sur une séquence, le même transposon est présent deux fois, dans le même sens. Lors de la recombinaison, les deux transposons s'apparient et il y a délétion du segment entre-eux (le segment B). b) sur une séquence, le même transposon est présent deux fois mais en sens opposé. Lors de la recombinaison, il y a échange est le segment entre les deux transposons se trouve renversé.*

Chapitre 2

Comparaison de séquences et distances entre séquences génétiques

Nous abordons dans ce chapitre la notion de quantification et de qualification de la ressemblance entre séquences. Il ne s'agit pas d'une description exhaustive des outils et méthodes pour la comparaison de séquences mais plutôt d'une présentation des principales idées sur lesquelles sont basées ces méthodes.

Dans un premier temps, nous décrivons les techniques de comparaison de séquences et en particulier l'alignement puisqu'il est aujourd'hui le moyen de comparaison le plus utilisé. Nous n'oublions pas les méthodes par réarrangements de segments qui fournissent un nouvel outil pour la comparaison de chromosomes ou de génomes entiers en comparant l'ordre respectif des gènes dans deux séquences. Dans un second temps nous nous attachons à décrire la définition de distances entre séquences servant à la construction d'arbres phylogénétiques. En particulier nous passons en revue un certain nombre de modèles de l'évolution à partir desquels un indice de similarité est dérivé en mesure de distance.

2.1 Méthodes pour la comparaison de séquences

Parmi les différents traitements que l'on peut effectuer sur des séquences génétiques, l'un des plus fréquemment réalisé est sans aucun doute la comparaison de deux ou plusieurs d'entre elles. Comme nous l'avons vu au chapitre précédent, les séquences portent le message génétique. Comparer des séquences permet de les étudier.

Lorsque le biologiste dispose d'une séquence inconnue, son but est d'identifier les gènes, les parties codantes et non codantes, puis de caractériser les fonctions de ces gènes. En se basant sur l'idée qu'*a priori* deux séquences qui se ressemblent codent pour les mêmes fonctions, comparer des séquences pour les étudier semble tout à fait naturel.

L'opération de comparaison peut être définie de manière très schématique comme l'identification des parties similaires des séquences étudiées. Si le concept est simple, sa mise en oeuvre l'est beaucoup moins car il faut définir formellement un critère de ressemblance.

Les premières méthodes de comparaison de séquences dont nous parlerons dans cette section sont les méthodes d'alignement. L'idée de l'alignement peut être résumée en utilisant l'image du

collier de perles [Sag96]. Imaginons que deux séquences génétiques soient des colliers dont chaque résidu est représenté par une perle. Le but de l'alignement est de mettre en correspondance un nombre maximal de perles. En supposant que le fil sur lequel sont enfilées les perles est légèrement plus long que la longueur obtenue en mettant les perles bout à bout, on peut réaliser cette mise en correspondance en glissant des perles ou des groupes de perles.

Les secondes méthodes dont nous parlerons sont les méthodes par réarrangement. Cette fois, nous disposons d'un ensemble de segments, numérotés. La comparaison porte sur l'ordre dans lequel apparaissent ces segments. On recherche alors, grâce à des opérations de remaniements de segments tels que celles dont il a été question à la fin du chapitre précédent, le nombre minimal d'opérations nécessaires pour transformer une permutation de ces segments, représentant un ordre dans une séquence, en une autre permutation, représentant l'ordre des mêmes segments dans une autre séquence.

2.1.1 Notations

Un **alphabet** est un ensemble fini de **lettres** (également nommés **symboles**). Un **mot** ou un **texte** écrit sur un alphabet A est une suite de lettres prises dans l'alphabet A. La **longueur** du texte u est le nombre de lettres qui le composent, on la note $|u|$. Le **mot vide** est le mot ne contenant aucune lettre, il est noté ε. Chaque mot u peut être indicé de 1 à $|u|$, on notera $u[i]$ la i-ième lettre de u.

Un **segment** d'un mot u est une sous-suite contiguë de lettres de u. $u[p..q]$, $1 \leq p \leq q \leq |u|$ (usuellement appelé facteur), représente le segment $u[p]u[p+1]\ldots u[q]$. $u[p,l]$, $1 \leq p \leq p+l \leq |u|$, représente $u[p]u[p+1]\ldots u[p+l-1]$.

L'opération de **concaténation** consiste en la mise bout à bout de deux mots. Si u et v sont deux mots, uv est la concaténation de u et v. L'opérateur de concaténation est représenté par un point (voir l'exemple ci-dessous).

Exemple 2.1 *Soit* $u =$`ATCCGTGACGGTAGCG`, $|u| = 16$, $u[1] =$ `A`, $u[2] =$ `T`, ... , $u[|u|] =$ `G`. $u[2..5] =$ `TCCG` *et* $u[2,4] =$ `TCCG`.

2.1.2 Différentes manières de comparer

Dès que l'on parle de comparaison d'objets, on pense immédiatement à la comparaison de deux objets de manière globale. C'est-à-dire que l'on va s'intéresser à la ressemblance des deux objets dans leur totalité. Il y a pourtant différentes manières d'aborder la comparaison de séquences. On choisira l'une ou l'autre façon de faire selon ce que l'on cherche ou ce que l'on désire montrer.

Comparaison globale ou locale.

La manière classique de procéder pour la comparaison de séquences est d'observer globalement les similitudes entre les séquences, c'est-à-dire observer les séquences dans leur totalité. Ce système de comparaison fonctionne correctement tant que les séquences que l'on compare ont de bonnes raisons de se ressembler dans leur totalité. Dès lors que l'on veut comparer des séquences qui ne sont plus suffisamment proches, une méthode se basant sur **la ressemblance globale**

ne permettra pas d'obtenir une information pertinente. En effet, si l'on compare deux séquences relativement éloignées de manière globale, la méthode risque de forcer la mise en correspondance de parties des deux séquences pour lesquelles il n'existe pas de similitude forte. Un autre processus de comparaison consiste à rechercher des portions des deux séquences ayant une bonne similitude. Il s'agit alors d'identifier **la ressemblance locale** entre les deux séquences. De telles méthodes offrent souvent plusieurs résultats auxquels est attaché un score reflétant l'intérêt ou la qualité de la ressemblance locale. Peu de méthodes offrent par contre la possibilité d'effectuer une recherche d'un meilleur sous-ensemble des ressemblances locales (ce qui reviendrait à fournir une ressemblance globale basée sur l'identification de ressemblances locales).

Comparaison deux à deux ou multiple.

La **comparaison multiple** consiste, lorsque l'on dispose d'un ensemble de n séquences à comparer directement les n séquences plutôt que d'effectuer $\frac{n(n-1)}{2}$ comparaisons deux à deux. Si la comparaison deux à deux est utile lorsque l'on cherche à établir une relation entre deux séquences, lorsqu'on recherche expressément des parties communes entre deux séquences, la comparaison multiple offre le plus souvent un résultat plus robuste et plus significatif. En effet, lorsqu'on compare des séquences deux à deux, il est impossible, du fait de l'indépendance des comparaisons, de mettre en relation les similitudes trouvées entre les séquences i et j et celles trouvées entre les séquences j et k. L'intérêt de la comparaison multiple est la mise en évidence d'une relation entre plusieurs séquences i,j,k, c'est-à-dire d'informations communes à i, j et k et non plus à (i et j) et à (j et k) comme le permettent les comparaisons deux à deux. D'autre part lorsqu'on détecte une portion similaire entre plus de deux séquences, celle-ci a clairement plus de chances d'être significative que la détection d'une portion similaire entre uniquement deux séquences. Même si la comparaison multiple est considérablement plus complexe que la comparaison deux à deux, il est des cas où elle est indispensable. Ce sont les cas où les informations communes à deux séquences forment un signal trop faible. Imaginons que nous disposions d'un ensemble de séquences protéiques codant pour la même fonction et que nous recherchions une sorte de signature de cette fonction, la comparaison multiple peut mettre en avant une ressemblance significative entre toutes les séquences alors qu'une comparaison deux à deux n'aurait peut être pas permis de la mettre en évidence. Les méthodes de reconstruction phylogénétique se basent le plus souvent sur une comparaison multiple de séquences pour inférer un arbre phylogénétique.

Comparaison directe ou indirecte

Une troisième distinction doit être faite entre les méthodes comparant directement les séquences entre elles et celles comparant les séquences à des objets externes. Un objet externe peut par exemple être un modèle, une expression régulière, un motif. Un tel mot sera une description de ce à quoi doivent ressembler les segments des séquences avec lesquels on veut identifier une ressemblance. Par exemple, la recherche des sites de fixation d'un leucine zipper peut se faire à partir de la recherche d'un des quatre mots `TGAGTCA`, `TGAGTAA`, `TGACTCA` et `TGACTAA` qui correspondent aux différents segments reconnus comme étant des sites de fixation. Le mot générique permettant la reconnaissance de ces quatre mots est $\texttt{TGA}^{\texttt{G}}/_{\texttt{C}}\texttt{T}^{\texttt{C}}/_{\texttt{A}}\texttt{A}$[2]. De manière générale, un mot générique est mot référence qui fait qu'il existe une certaine relation de similarité entre ce mot et un ensemble de mots similaires. Dans le cas des comparaisons directes, il est également

2. $\texttt{G}/_{\texttt{C}}$ signifie que le mot contient à cet emplacement soit un `G`, soit un `C`.

possible d'établir de tels ensembles de mots similaires et un mot générique résumant en quelque sorte cet ensemble. Mais il faut bien avoir à l'esprit que dans ce cas le mot générique est obtenu après comparaison des séquences et identification des ensembles de mots similaires. Dans le cadre de la comparaison indirecte, le mot générique est construit au fur et à mesure de l'identification de mots similaires [Sag96].

2.1.3 La matrice de points

Nous décrivons ici une première méthode, simple, permettant la comparaison qualitative de séquences possédant une bonne similitude. Celle-ci est basée sur une représentation matricielle des zones de ressemblance entre deux séquences. Les deux séquences à comparer sont écrites sur les entêtes de colonne et de ligne (voir figure 2.2), et l'on place un point dans la matrice si les deux résidus des séquences sont identiques.

Si les deux séquences sont identiques, on observe une ligne de points sur la diagonale de la matrice (celle qui traverse la matrice du coin en haut à gauche au coin en bas à droite). Si les deux séquences sont similaires sauf à quelques substitutions près, la majorité des cases de la diagonale possède un point. Enfin, si il existe des insertions–délétions, on peut voir apparaître de petites diagonales parallèles à la diagonale principale. Dans le cas où les séquences possèdent des similitudes locales, des diagonales peuvent apparaître n'importe où dans la matrice. Des diagonales apparaissant perpendiculairement à la diagonale principale indiquent la détection d'une partie dont le sens de lecture est inversé (voir figure 2.1).

La comparaison directe de deux séquences en plaçant un point à chaque fois que deux résidus sont trouvés identiques aboutit le plus souvent à une image floue, très noircie. Cela est dû à la petite taille de l'alphabet : plus l'alphabet est petit, et plus l'équiprobabilité de rencontrer une lettre est grande. Il y a donc génération d'un bruit qui nuit à l'identification de parties communes. Lorsque nous travaillons avec des séquences ADN, la comparaison se fait rarement point par point mais on compare le plus souvent des groupes consécutifs de plusieurs bases (2, 3 ou 4). Chaque point de la matrice représente alors deux petits segments identiques de la séquence. Cela permet d'augmenter la lisibilité des points et d'améliorer la détection des diagonales en supprimant une partie du bruit.

La difficulté de la comparaison consiste ensuite à choisir lesquelles de ces diagonales sont optimales. Ce choix des diagonales est réalisé par les méthodes d'alignement au travers de l'optimisation d'une fonction de score.

2.1.4 Alignement de séquences

Le but de cette section n'est pas de décrire complètement les méthodes d'alignement qui existent actuellement. Nous nous intéressons ici aux fondements et aux principales techniques utilisées lorsqu'il s'agit de calculer un alignement. Le lecteur désireux d'en connaître plus pourra se reporter à [Wat95, SM97, Sag96]. Aussi nous omettrons les discussions relatives au choix des meilleurs paramètres du calcul de la fonction de score associée à un alignement.

Un alignement permet de localiser et de mesurer la ressemblance entre deux séquences : d'une part l'alignement montre les zones de ressemblance, d'autre part le score d'alignement quantifie la ressemblance.

Dans un premier temps, nous décrivons le principe de l'alignement puis la notion de score

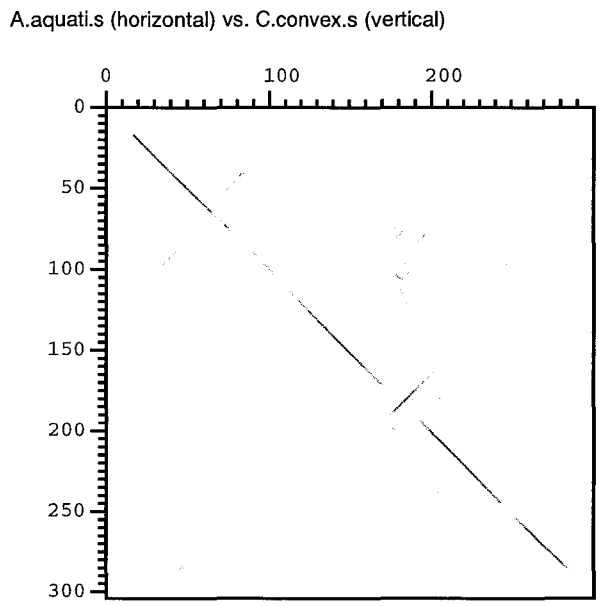


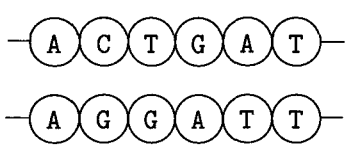
FIG. 2.1 – *Exemple de matrice de points. Les séquences mesurent environ 300pb. Les diagonales représentent les segments identifiés. Les séquences étudiées ici sont très proches. On détecte une partie dont le sens de lecture est renversé au centre.*

associé à un alignement. Ensuite, nous décrivons la méthode classique de programmation dynamique permettant un calcul efficace d'un score d'alignement, et du schéma d'alignement associé.

Principe de l'alignement

Comme nous l'avons vu avec l'image du collier de perles, un alignement est la recherche de mise en correspondance des lettres d'une séquence par rapport à une autre en respectant l'ordre des lettres dans les séquences. Ce dernier point est très important. L'image du collier permet d'ailleurs une excellente représentation : on peut faire glisser les perles, ou des groupes de perles, mais il n'est jamais possible d'intervertir deux groupes de perles. L'ordre global d'apparition des résidus dans les séquences doit impérativement être conservé. Nous appelons cet ordre, **ordre naturel des séquences**.

Soient les deux séquences $u =$ `ATCGAT` et $v =$ `AGGATT`. Une représentation d'un alignement (avec la vue du collier de perles) est la suivante :



Une autre représentation, plus habituelle, est :

$$\begin{array}{ll} u = & \texttt{AtcgaT} \\ & \texttt{:\ \ :} \\ v = & \texttt{AggatT} \end{array}$$

où les ':' indiquent les lettres identiques et les minuscules les mutations ponctuelles. On dira

alors qu'il y a eu une **substitution** de `t` en `g`, une autre de `c` en `g`, etc. En anglais on dira **mismatch**. Lorsque les bases sont identiques pour une même position, on parlera de **match**. Cela définit ce que nous appelons un **script d'édition** de u en v. Un alignement n'est autre que la représentation du script d'édition. D'autres scripts d'édition entre u et v peuvent être envisagés si l'on étend le nombre d'opérations autorisées dans le script d'édition:

```
u =  ActGA-T          u =  ActGAT-
     :  ::  :               :  :::
v =  Ag-GAtT          v =  Ag-GATt
```

"-" représente alors une **insertion** (ou une **délétion**, suivant la lecture que l'on en a) dans l'une des deux séquences. On parlera également d'**indel** ou en anglais de **gap**. Si la transformation est de u vers v, le script d'édition du dernier exemple sera: consevation de `A`, substitution de `C` en `G`, délétion de `T`, conservation de `G`, `A` et `T` puis insertion de `T`. Ces mêmes alignements peuvent être retrouvés sur la matrice de points (voir figure 2.2).

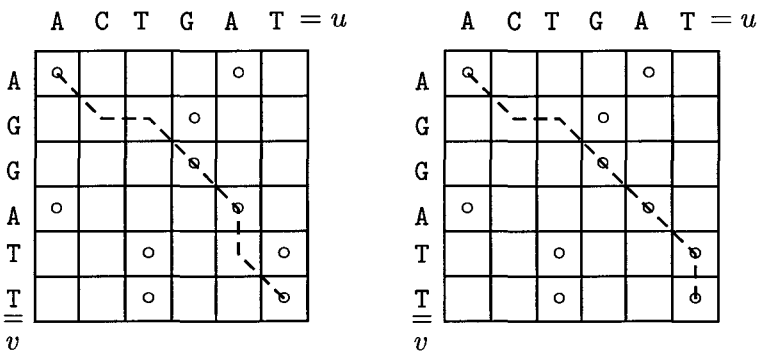


FIG. 2.2 – *Exemples de matrices de points (dot-matrix).*

Notons que les séquences à aligner n'ont pas forcément la même longueur, que les séquences alignées, c'est-à-dire réécrites avec '-', ont une longueur supérieure ou égale à la plus grande des deux, et que l'alphabet sur lequel sont écrites les séquences alignées est la réunion des alphabets des deux séquences et de la lettre "-".

L'alignement doit refléter les meilleures similitudes entre les séquences. Pour ce faire, on attribue un **score** à chacune des mises en correspondance possibles. Le problème de la recherche d'un alignement devient donc un problème d'optimisation d'une fonction de score d'alignement.

Alignement local. Le problème de l'alignement global est un sous problème de l'alignement local. Pour ce dernier, il ne s'agit plus de trouver une concordance lettre à lettre globale des deux séquences mais de trouver les parties des deux séquences qui s'alignent le mieux. Le but est donc de trouver les indices p,q,r,s tels que les deux sous-séquences $u[p,q]$ et $v[r,s]$ ont un score d'alignement global maximal. Intuitivement, l'alignement local est une méthode permettant de repérer les meilleures diagonales sur la représentation de la matrice de point.

Alignement multiple. On parle d'alignement multiple lorsque l'on désire aligner plus de deux séquences. L'alignement multiple permet de repérer des parties bien conservées entre toutes les séquences mises en jeu. Il est notamment utilisé à des fins de reconstruction phylogénétique.

Les techniques mises en oeuvre pour les calculs d'alignement multiples sont différentes de celles relatives à l'alignement global et local. Nous ne les décrivons pas ici car elles sont hors de notre propos, nous renvoyons le lecteur à [Wat95, SM97, Sag96] pour un aperçu de ces méthodes.

Score entre séquences

Les séquences à aligner sont notées u et v et leurs longueurs respectives m et n. Nous noterons respectivement s_i la i-ème lettre d'une séquence s et s'_i la i-ème lettre de la séquence s alignée.

Définition du score On attribue aux opérations une fonction de coût élémentaire (on parlera également de poids ou de score). Notons ω cette fonction. Ce coût élémentaire correspond au score associé à chacune des positions des séquences lorsqu'elles sont représentées l'une sous l'autre (voir figure 2.3). On distingue le coût associé aux mutations du coût associé aux insertions/délétions. Le premier est usuellement donné par une **matrice de substitution** (ou matrice de score) σ qui représente les coûts de tous les appariements possibles (voir deux exemples figure 2.4). Le second est fixé par une constante positive g. La fonction ω est définie de la manière suivante :

$$\omega_{u'_i,v'_j} := \begin{cases} \sigma(u'_i,v'_j) & \text{si } u'_i \neq - \text{ et } v'_j \neq - \\ \text{-g} & \text{si } u'_i = - \text{ ou } v'_j = - \\ 0 & \text{sinon} \end{cases}$$

Le score d'un alignement entre u et v peut être ainsi défini :

$$score(u,v) := \sum_{i=1}^{l} \omega_{u'_i,v'_i}$$

où l est la longueur de l'alignement.

```
AGAggtaggCTTT-TTATTaCacAAgTTAAcTTgAcatAAaAgTtAAAAgGcTtTTATaT
:::      :::: ::::: :  :: :::: :: :   :: : : :::: : : :::: :
AGAtaataaCTTTcTTATTtCtgAA-TTAAaTTaAttaAAgAaTaAAAAaGtTcTTATtT
```

FIG. 2.3 – *Représentation d'un alignement de deux séquences.*

	A	T	C	G
A	1	0	0	0
T	0	1	0	0
C	0	0	1	0
G	0	0	0	1

a)

	A	T	C	G
A	3	0	0	1
T	0	3	1	0
C	0	1	3	0
G	1	0	0	3

b)

FIG. 2.4 – *Exemples de matrices de score pour l'ADN. a) matrice unitaire. b) matrice qui différencie les transitions des transversions.*

Fonction de pénalisation. La prise en compte des insertions/délétions de manière linéaire n'est pas proche de la réalité biologique. En fait, une suite d'insertions (respectivement, de délétions) unitaires adjacentes dans l'évolution entre deux séquences homologues est rare. Une telle suite provient plus probablement d'un seul événement d'insertion (respectivement de délétion) multiple. Si l'on pénalise une telle insertion (respectivement délétion) multiple de manière linéaire en fonction du nombre de caractères insérés (respectivement délétés), le coût la rend prohibitive, alors qu'elle est probable dans l'évolution. Au lieu d'une fonction de pénalisation linéaire, on utilise une fonction concave, la plupart du temps une fonction affine : $g(k) = g_o + g_e \times (k-1)$, avec $g_o \geq g_e$, et où g_o est la pénalisation d'ouverture de la zone d'indels, g_e, celle de son extension d'un caractère, et k sa longueur.

Méthodes pour l'alignement

Les algorithmes décrits ici considèrent une fonction de pénalisation linéaire pour les zones d'indels.

L'algorithme de Needleman et Wunsch [NW70]. Needleman et Wunsch furent les premiers à proposer un algorithme de calcul d'un alignement global de deux séquences génétiques (il s'agissait de protéines). L'algorithme est basé sur un concept de calcul dénommé **programmation dynamique**.

Le calcul est obtenu par maximisation d'une fonction de score $S_{i,j}$ définie de manière récursive:

$$S_{i,j} = \max \left\{ \begin{array}{c} S_{i-1,j-1} + \omega(u_i,v_j), \\ S_{i-1,j} - g, \\ S_{i,j-1} - g \end{array} \right\}$$

avec $S_{0,0} = 0$, $S_{0,j} = \sum_{k=1}^{j} \omega(-,v_j), 1 \leq j \leq n$, $S_{i,0} = \sum_{k=1}^{i} \omega(u_i,-), 1 \leq i \leq m$.

Les trois termes de la fonction de récurrence correspondent aux trois situations possibles : substitution, insertion, délétion. La preuve de ce résultat peut être trouvée dans [Wat95].

Une **matrice de programmation dynamique** est obtenue par le calcul de tous les $S_{i,j}$. $S_{i,j}$ correspond au score d'alignement des préfixes $u[1..i]$ et $v[1..j]$. La valeur $S_{m,n}$ donne le score de l'alignement. L'alignement peut ensuite être déduit par remontée de la matrice. Il suffit de se souvenir lors du calcul de la matrice d'où provient le résultat de chaque case (haut, diagonale, gauche). Un exemple de calcul est donné à la figure 2.5.

L'algorithme de Smith et Waterman [SW81]. Cet algorithme est une version modifiée du précédent pour le calcul du meilleur alignement local. Pour connaître le meilleur alignement local, il suffit de prendre dans la matrice de programmation dynamique la case de valeur la plus élevée.

Le calcul de la matrice de programmation dynamique est obtenu par le calcul d'une fonction de score récursive $S'_{i,j}$ définie ainsi:

$$S'_{i,j} = \max \left\{ \begin{array}{c} S'_{i-1,j-1} + \omega(u_i,v_j), \\ S'_{i-1,j} - g, \\ S'_{i,j-1} - g, \\ 0 \end{array} \right\}$$

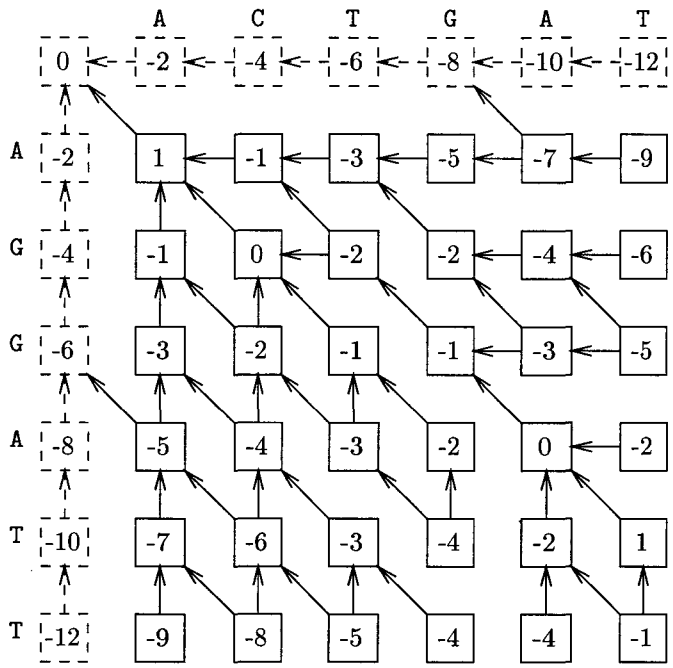


FIG. 2.5 – *Exemple de calcul d'une matrice de programmation dynamique avec l'algorithme de Needleman et Wunsch. Le score d'un match est de* 1*, le score d'un mismatch* -1 *et la pénalité associée aux gaps est* -2*. Chaque carré représente une case de la matrice, les flèches indiquent le schéma de remontée permettant de trouver l'alignement correspondant au score calculé.*

avec $S'_{0,0} = 0$, $S'_{0,j} = 0 \quad 1 \leq j \leq m$, $S'_{i,0} = 0 \quad 1 \leq i \leq n$.

Des algorithmes ont été proposés permettant le calcul de tous les alignements locaux [WE87]. Ils demandent un recalcul partiel de la matrice. On trouve également des algorithmes capables de détecter dans le même temps les segments similaires et les segments similaires inversés [Wat95].

2.1.5 BLAST et FASTA

BLAST (Basic Local Alignement Search Tool) et FASTA sont deux programmes permettant la recherche de séquences dans les bases de données. Le problème qu'ils traitent est le suivant : étant donnée une séquence A, rechercher l'ensemble des séquences similaires à cette séquence dans une base de donnée. Ce type d'outil est fondamental pour le biologiste, même s'il s'agit d'algorithmes heuristiques car il permet d'accéder à une première sélection et d'étudier ensuite plus en profondeur les résultats obtenus grâce à des outils plus fins.

BLAST [AGM+90]

BLAST permet l'identification des meilleurs segments similaires entre deux séquences. Une paire de segments est un couple de segments, de même longueur, pas nécessairement identiques, et pris dans l'une et l'autre des séquences. BLAST affecte à une paire de segments le score de leur alignement sans insertion/délétion. Les poids des substitutions sont donnés par des matrices.

BLAST fonctionne de la manière suivante. BLAST recherche les paires de segments de scores

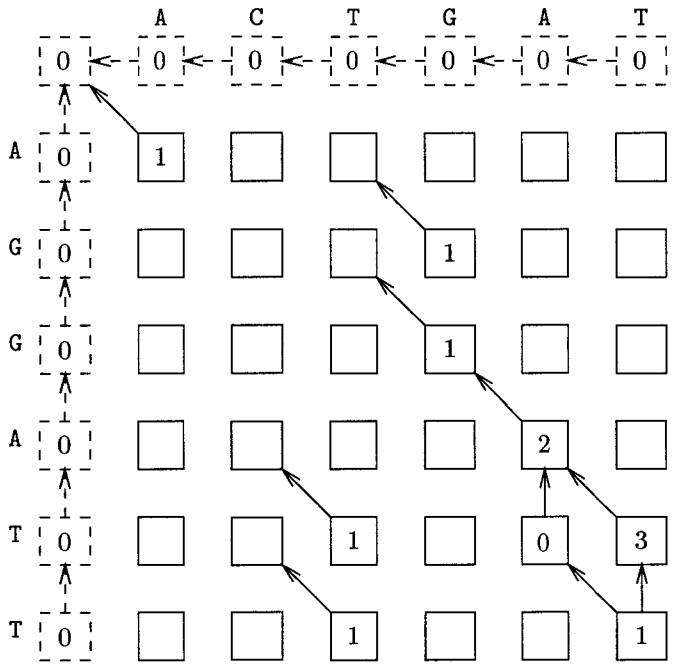


FIG. 2.6 – *Exemple de calcul d'un alignement local maximal avec l'algorithme de Smith et Waterman. Les scores sont les mêmes que pour la figure 2.5. Les cases vides contiennent 0. L'alignement local maximal a un score de 3, il correspond à GAT/GAT*

maximaux entre une séquence requête et chaque séquence de la base cible. Une valeur de seuil, fixée par l'utilisateur, permet de n'extraire que les paires de segments ayant un score significatif.

La recherche des paires de segments est basée sur la recherche de paires de segments de plus petite longueur, fixée, ce qui permet d'utiliser des algorithmes très rapides. Une fois ces petites paires exhibées et sélection faite d'un sous-ensemble par une valeur de seuil relative à leur score, celles-ci sont étendues à droite et à gauche pour tenter de trouver une paire de longueur supérieure.

FASTA [LP85, PL88]

FASTA est quand à lui basé sur la recherche de diagonales dans la matrice d'alignement local. Si B représente une séquence de la base de donnée et A la séquence requête, FASTA définit une paire comme un couple de positions (i,j) telle qu'il existe deux segments identiques de longueur k, l'un dans la séquence A et l'autre dans la séquence B. Le paramètre k est fixé par avance. Pour chaque séquence B de la base de données, FASTA calcule le nombre de paires de chaque diagonale de la matrice de points entre A et B.

Une fois le nombre de paires comptées, FASTA juge une séquence B similaire à la séquence A si il existe une ou plusieurs diagonales possédant un nombre suffisant de paires.

La rapidité de FASTA provient du fait que chaque paire est codée par un entier. C'est une procédure classique que de faire cette tranformation. Si le segment s'écrit a_k,a_{k-1},... ,a_1, alors son code sera $\sum_{i=1} ka_i|\sigma|^i$, avec σ l'alphabet sur lequel est écrit le segment. Le calcul des décompte se fait alors en temps linéaire par rapport au nombre de paires. Un alignement est

ensuite produit en utilisant les diagonales sélectionnées.

2.1.6 DIALIGN: alignement basé sur des segments

DIALIGN est une méthode proposée par Morgenstern et *al.* [MDW96, MFDW98, LMR99]. Cette méthode introduit une nouvelle façon de définir l'alignement. DIALIGN ne base pas un alignement sur la mise en concordance de résidus seuls mais sur la mise en concordance de segments complets similaires, dénommés **diagonales**. La dissimilarité est limitée aux seules substitutions.

L'alignement obtenu est le résultat du choix d'un sous-ensemble de diagonales parmi l'ensemble de toutes les diagonales. Ce choix est fait de manière assez classique au travers de la maximisation d'un score d'alignement. Ce score est défini comme la somme des poids de chaque diagonale:

$$S_{\text{DIALIGN}} = \sum_{D \in \mathcal{A}} w(D)$$

Nous pouvons déjà noter que le score de l'alignement ne dépend pas des parties qui se trouvent entre chaque diagonale. Ainsi, si deux diagonales sont éloignées ou si elles sont proches, le score d'alignement est le même.

Le poids des diagonales est défini de la manière suivante. Soit D une diagonale de longueur l possédant m résidus identiques. Notons $P(l,m)$ la probabilité pour une diagonale de longueur l de contenir m résidus identiques,

$$P(l,m) = \sum_{i=m}^{l} \binom{l}{i} p^i (1-p)^{l-i}$$

où p représente la probabilité que deux bases soient identiques (dans la cas d'une distribution uniforme, $p = 0.25$ pour les acides nucléiques et $p = 0.05$ pour les acides aminés). On définit ensuite la quantité $E(l,m) := -\ln P(l,m)$ puis le poids de la diagonale D:

$$w(D) = \begin{cases} E(l,m) & \text{si } E(l,m) > T \\ 0 & \text{sinon} \end{cases}$$

où T représente une valeur de seuil positive ou nulle.

Une fois défini le poids d'une diagonale, le score de l'alignement peut être calculé. Celui-ci est défini au travers d'un critère de consistance: un ensemble de diagonales est consistant si les diagonales ne se croisent pas et si un résidu d'une séquence n'appartient pas à deux diagonales d'une même autre séquence.

Pour l'alignement multiple, le poids w des diagonales est modifié afin de prendre en compte le nombre de séquences couvertes par une diagonale et ainsi favoriser le critère de multiplicité.

2.1.7 Méthodes par réarrangement de segments

Les méthodes dont nous allons discuter dans cette section sont différentes de celles envisagées dans les sections précédentes. Elles concernent cette fois des segments complets, le plus souvent

des gènes. La thèse de David Christie [Chr98] constitue une bonne introduction aux problèmes dont nous allons discuter.

Nous avons vu dans le chapitre précédent qu'il existait des mutations des séquences d'un autre ordre de grandeur que celui des mutations ponctuelles : les réarrangements. Si les méthodes décrites jusqu'à maintenant pouvaient se révéler satisfaisantes lorsqu'on se situait au niveau d'un gène ou d'une protéine, il n'en est plus de même dès lors que l'on passe au niveau du chromosome ou du génome. On désire comparer des génomes en observant les déplacements de certaines parties de leur séquence. Ce type d'analyse n'est pas nouveau puisqu'à la fin des années 30 Dobzhansky et Sturtevant [SD36] présentaient un scénario de réarrangements pour différentes espèces de Drosophile. Cependant, la recherche de méthodes permettant de résoudre le problème du calcul du nombre minimum de réarrangements permettant de réordonner un génome en un autre est assez récente.

L'intérêt de l'étude des réarrangements génomiques est qu'il semble être possible d'inférer des relations phylogénétiques de manière plus précise. Cela permet d'étudier les relations entre espèces ayant divergé depuis plus longtemps. En effet, un gène peut être reconnu par sa fonction ou par sa protéine, même s'il est différent au niveau de la séquence ADN. Par conséquent, il est possible de comparer l'ordre des gènes dans deux espèces différentes, même si elles ont divergé depuis longtemps. L'utilisation de réarrangements génomiques à des fins phylogénétiques a été introduite par Sankoff [SLA+92].

Il ne s'agit donc plus d'identifier des parties communes entre deux génomes, mais, ces parties communes étant données, et connaissant leur ordre dans les deux génomes, le but est de trouver le nombre minimum d'opérations de réarrangements de ces segments permettant le passage d'un génome à l'autre.

Nous énumérons ici les différents problèmes qui ont été étudiés. La plupart de ces problèmes de réarrangement ont été montrés NP-durs. Des heuristiques ont donc été mises au point pour aboutir à une solution avec un ratio de garantie. Nous ne détaillons pas ici les algorithmes de résolution, ils sont complexes. On pourra trouver des détails dans [Han95b, Gus97, SM97].

Dans la suite, l'ordre des gènes dans deux génomes sera représenté par les permutations $\pi = (\pi_1 ... \pi_n)$ et $\sigma = (\sigma_1 ... \sigma_n)$. Pour tout gène x, on note $-x$ son inverse.

Réarrangement par inversions

Une **inversion** $\rho(i,j)$ d'un intervalle $[i,j]$ est le réarrangement :

$$\begin{pmatrix} 1 & 2 & \dots & i-1 & \boxed{i} & \boxed{i+1} & \dots & \boxed{j-1} & \boxed{j} & j+1 & \dots & n \\ 1 & 2 & \dots & i-1 & \boxed{j} & \boxed{j-1} & \dots & \boxed{i+1} & \boxed{i} & j+1 & \dots & n \end{pmatrix}$$

qui renverse l'ordre des gènes $\pi_i \pi_{i+1} \dots \pi_j$ et transforme donc $(\pi_1 \dots \pi_{i-1} \pi_i \dots \pi_j \pi_{j+1} \dots \pi_n)$ en $(\pi_1 \dots \pi_{i-1} \pi_j \dots \pi_i \pi_{j+1} \dots \pi_n)$.

Le **problème de la distance par inversions** (*"reversal distance problem"*, ou *"sorting by reversals"* en anglais) s'énonce ainsi : étant données deux réarrangements π et σ, trouver une série d'inversions $\rho_1, \rho_2, \dots, \rho_t$ telles que $\pi \cdot \rho_1 \cdot \rho_2 \cdot \dots \cdot \rho_t = \sigma$ et tel que t soit minimum.

Le problème peut être spécifié, soit pour des inversions signées (ou orientées), soit pour des inversions non signées. Dans le cas des inversions orientées, chaque gène possède un sens de

lecture qui peut être différent dans les deux permutations. On représentera alors un gène devancé par un signe: + si il se lit de 3' vers 5' et − sinon. Par conséquent l'inverse de la permutation $(+1, -2, +3)$ sera $(-3, +2, -1)$. Cela ajoute une contrainte supplémentaire au problème.

La figure 2.7 montre l'exemple de la transformation de B.oleracea (chou) en B.campestris (navet).

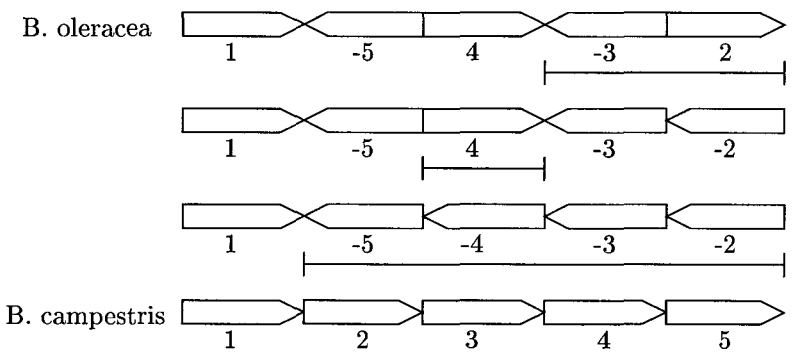


FIG. 2.7 – *Transformation du chou (B. oleracea) en navet (B. campestris) grâce à trois inversions [Han95a]*

Les premiers à obtenir des résultats sur le calcul de la distance par inversions furent Kececioglu et Sankoff [KS93]. Ils développèrent la première heuristique pour résoudre le problème, pour des inversions non signées, garantissant de trouver un nombre de d'inversions n'excédant pas le double du nombre minimal d'inversions avec une complexité en temps en $O(n^2)$. Bafna et Pevzner [BP93] présentèrent ensuite un algorithme garantissant une approximation de $\frac{7}{4}$ de la solution optimale dans le cas non signé et de $\frac{3}{2}$ dans le cas signé. Hannenhalli et Pevzner [HP95, BH96] ont découvert un algorithme polynomial, en $O(n^4)$, capable de résoudre le problème pour des inversions signées. Le cas signé se révèle donc plus facile que le cas non signé, montré NP-dur [Cap97].

Réarrangement par transpositions

Une **transposition** $\rho(i,j,k)$ (avec $k \notin [i,j]$) est le réarrangement :

$$\begin{pmatrix} 1 & \dots & i-1 & i & \dots & j-1 & j & \dots & k-1 & k & \dots & n \\ 1 & \dots & i-1 & j & \dots & k-1 & i & \dots & j-1 & k & \dots & n \end{pmatrix}$$

qui déplace une suite de gènes de π à un autre emplacement. Il est à noter que si $i < j < k$ alors $\pi \cdot \rho(i,j,k)$ échange les blocs de gènes $\pi_i \dots \pi_{j-1}$ et $\pi_j \dots \pi_{k-1}$.

Le **problème de la distance par transpositions** (*"sorting by transpositions"* en anglais) s'énonce de manière similaire à celui de la distance par inversions : étant données deux réarrangements π et σ, trouver une série de transpositions $\rho_1, \rho_2, \dots, \rho_t$ telles que $\pi \cdot \rho_1 \cdot \rho_2 \dots \rho_t = \sigma$ et tel que t soit minimum.

La mise en oeuvre de méthodes permettant de traiter ce problème fait suite au problème de la distance par inversions. Le problème se révèle en fait être plus dur que le problème de la distance par inversions [BP95]. Il est décrit dans cet article deux algorithmes heuristiques garantissant l'un une approximation de 1.75 et l'autre une approximation de 1.5 de la solution optimale.

Réarrangement par translocations

L'opération de translocation est plus complexe que les deux opérations précédentes. Non seulement elle fait intervenir l'ordre des gènes mais également la position de ces gènes sur les chromosomes.

L'opération de translocation échange deux parties de deux chromosomes. On distingue deux types de translocations. Si l'on dispose de deux chromosomes X et Y tels que $X = X_1X_2$ et $Y = Y_1Y_2$, une **translocation préfixe-préfixe** a pour résultat $X = X_1Y_2$ et $Y = Y_1X_2$; une **translocation préfixe-suffixe** a pour résultat $X = X_1 - Y_1$ et $Y = -Y2X2$.

Une **translocation** $\rho(X,Y,i,j)$ sur une paire de chromosomes $X = (x_1,x_2,\ldots,x_m)$ et $Y = (y_1,y_2,\ldots,y_n)$ divise le chromosome X entre les gènes x_{i-1} et x_i et le chromosome Y entre les gènes y_{j-1} et y_j.

Une **translocation préfixe-préfixe** $\rho_{pp}(X,Y,i,j)$ a pour résultat les deux chromosomes $(x_1,\ldots,x_{i-1},y_j,\ldots,y_n)$ et $(y_1,\ldots,y_{j-1},x_i,\ldots,x_m)$.

Une **translocation préfixe-suffixe** $\rho_{ps}(X,Y,i,j)$ a pour résultat les deux chromosomes $(x_1,\ldots,x_{i-1},-y_{j-1},\ldots,-y_1)$ et $(-x_m,\ldots,-x_i,y_j,\ldots,y_n)$.

Le problème est ici de trouver le nombre minimum de translocations nécessaires pour passer d'un génome A à un génome B représentés par l'ensemble de leurs chromosomes.

La figure 2.8 illustre les deux types de translocations envisagées ici, et la figure 2.9 représente un exemple d'évolution par translocation.

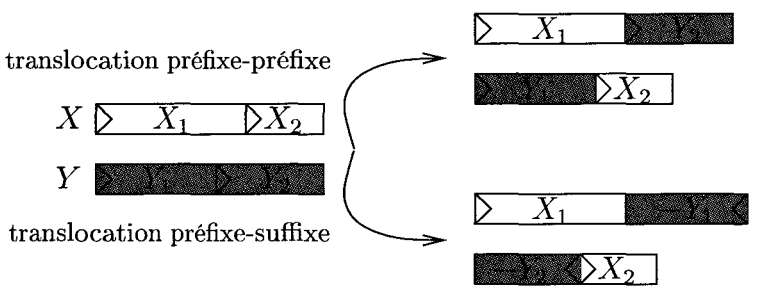


FIG. 2.8 – *Illustration des deux types de translocations (tiré de [Han95b]).*

Kececioglu et Ravi [KR95] ont proposé un algorithme permettant de calculer une distance par transpositions n'étant pas plus de deux fois plus grande que la distance optimale par transpositions. Ils montrent également un algorithme avec une approximation de 1.5 si l'on combine translocations et inversions. Hannenhalli [Han95a] montre un algorithme polynomial pour résoudre le problème dans le cas signé.

Duplication de génomes

Il existe un événement rare dans l'évolution qui est la duplication d'un génome. Au cours de la division cellulaire, une erreur fait que chaque chromosome, au lieu d'être présent une seule fois dans la cellule fille, est présent exactement deux fois. Des exemples de plus en plus nombreux montrent que cet événement est à l'origine de la naissance de certaines espèces, notamment des plantes. En particulier, il a été montré que la levure provenait très certainement d'un génome

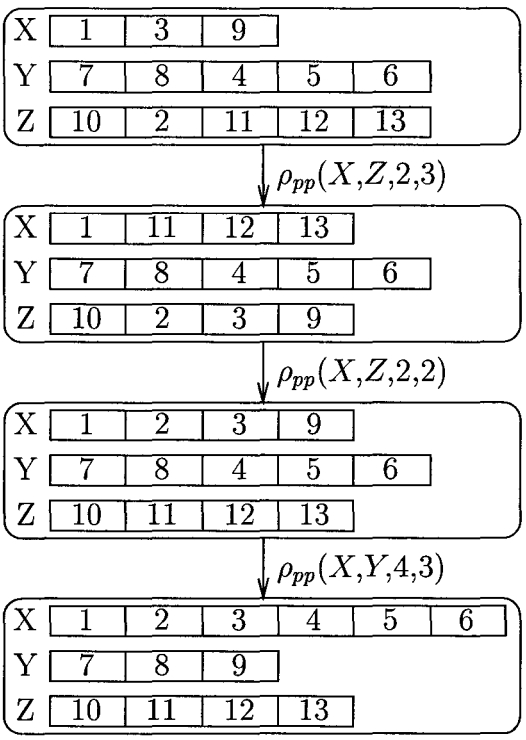


FIG. 2.9 – *Exemple d'evoluion par translocations d'un génome comportant trois chromosomes (tiré de [Han95b]).*

ancestral ayant subi un événement de duplication [WS97].

Le problème qui est posé est le suivant. Disposant d'un génome dont chaque gène est présent deux fois, nous désirons reconstruire un génome ancestral qui pourrait être à l'origine du génome étudié si l'on accepte qu'il y a eu un jour un événement de duplication de génome. Plus formellement, nous disposons d'un ensemble de $2n$ chromosomes, chaque chromosome est représenté par une liste de gènes et nous recherchons le nombre minimal de translocations permettant de passer d'un génome ancestral doublé, comportant deux fois les mêmes n chromosomes au génome étudié.

El-Mabrouk et *al.* proposent dans [EMBS99] un algorithme polynomial de calcul du nombre de translocations et du génome ancestral où l'on considère un génome comme un ensemble de chromosomes, et chaque chromosome comme une suite de gènes orienté. Chaque gène est bien sûr présent deux fois dans le génome dont on recherche l'ancêtre. La technique de résolution adoptée est dérivée de celle proposée par Hannenhalli dans le calcul de la distance par translocations [Han95a].

Distance synténique

Ferretti et *al.* [FNS96] ont proposé une mesure de distance basée sur les remaniements un peu plus abstraite que celles dont nous avons discuté précedemment. Dans ce cadre, un génome est considéré comme un ensemble de gènes dont on ne connaît pas les positions sur les chromosomes. Un chromosome est donc un ensemble non ordonné de gènes et un génome un ensemble de

chromosomes. Trois opérations permettant les remaniements de ces chromosomes sont possibles :

- l'échange de gènes entre deux chromosomes,
- la fission d'un chromosome en deux,
- la fusion de deux chromosomes en un seul.

Comme dans les cas précédents, la distance entre deux génomes correspond au nombre minimal de ces opérations nécessaires à la transformation d'un génome en un autre.

Ferreti et *al.* ont proposé un algorithme heuristique permettant un calcul approximatif de la distance synténique [FNS96]. DasGupta et *al.* [GJK+97] ont montré que le problème du calcul de la distance synténique était NP-dur. Ils décrivent également un algorithme garantissant un calcul approché de la distance au maximum égal à deux fois la distance minimale.

2.2 Distance et phylogénie

Dans ce chapitre nous abordons les différentes distances qui ont pu être définies pour mesurer la similarité entre des séquences. L'énumération que nous faisons n'est pas exhaustive mais passe en revue les principales notions de distance.

Ces distances servent en particulier à l'élaboration d'arbres phylogénétiques. Un arbre phylogénétique est la représentation des relations de filiation entre les espèces. On appelle problème de reconstruction phylogénétique, celui de calculer l'arbre phylogénétique. Il existe plusieurs approches à ce problème ; une dite "par distance" prend comme donnée une matrice des distances entre espèces. Parmi toutes les distances possibles [DT93, SOWH96], les distances calculées à partir de la comparaison de séquences tiennent maintenant une place de choix. A partir de cette matrice, on peut calculer un arbre qui approche au mieux les distances observées, c'est-à-dire, de la matrice. Nous renvoyons le lecteur à [SOWH96, Ber98, DT93] pour les méthodes de reconstruction d'arbres.

Il existe d'autres manières de recontruire des arbres phylogénétiques n'étant pas basées sur le calcul d'une matrice de distances. Le livre de Darlu et Tassy [DT93] consacré à l'ensemble des méthodes de reconstruction phylogénétique constitue une excellente introduction.

2.2.1 Notion de distance

Définition 2.1 *Un* ***indice de proximité*** *sur un ensemble E est une application f de $E \times E$ dans $\mathbb{R}$ telle que :*

- $f(i,j) = f(j,i)\ \forall (i,j) \in E \times E$
- $f(i,j)$ *est d'autant plus élevé que i est proche de j, pour tout* $(i,j) \in E \times E$

Définition 2.2 *Un* ***indice de similarité*** *sur un ensemble E est une application f de $E \times E$ dans $\mathbb{R}$ telle que :*

- $f(i,j) = f(j,i)\ \forall (i,j) \in E \times E$
- $\forall i \in E, f(i,i) = \max(f(k,j))\forall (k,j) \in E \times E$

Définition 2.3 *Un **indice de dissimilarité** sur un ensemble E est une application f de $E \times E$ dans $\mathbb{R}^+$ telle que :*

- $f(i,j) = f(j,i)\ \forall (i,j) \in E \times E$
- $\forall i \in E, f(i,i) = 0$

Définition 2.4 *Une **distance** sur un ensemble E est un indice de dissimilarité sur E vérifiant:*

- $f(i,j) = 0 \Longrightarrow i = j$
- $f(i,k) \leq f(i,j) + f(j,k)\ \forall i,j,k \in E$ *(inégalité triangulaire)*

Définition 2.5 *Une distance est dite **additive** si*

$$f(i,j) + f(k,l) \leq \max(f(i,k) + f(j,l), f(i,l) + f(j,k))$$

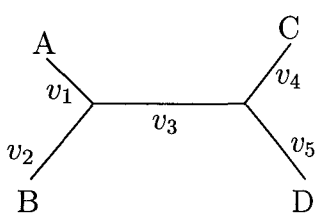
La condition nécessaire pour qu'une distance soit additive est aussi appelée **condition des quatre points**. Cela signifie que l'on peut représenter les quatres objets i,j,k,l sur un arbre non enraciné tel que la somme des longueurs des branches de cet arbre pour rejoindre a et b corresponde exactement à $f(a,b)$ (voir figure 2.10).

Définition 2.6 *Une distance est dite **ultramétrique** si*

$$f(i,k) \leq \max(f(i,j) + f(j,k))$$

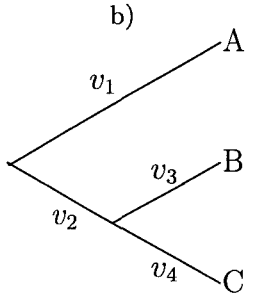
La propriété d'ultramétricité permet de contruire des arbres phylogénétiques pour lesquels la distance entre deux espèces correspond exactement à la somme des longueurs des branches qui les joignent. De plus, l'arbre peut être enraciné de façon à ce que toutes les espèces soient équidistantes de la racine (voir figure 2.10).

a)



Propriétés d'additivité :
$d_{AB} = v_1 + v_2$
$d_{AC} = v_1 + v_3 + v_4$
$d_{AD} = v_1 + v_3 + v_5$
$d_{BC} = v_2 + v_3 + v_4$
$d_{BD} = v_2 + v_3 + v_5$
$d_{CD} = v_4 + v_5$

b)



Propriétés d'additivité :
$d_{AB} = v_1 + v_2 + v_3$
$d_{AC} = v_1 + v_2 + v_4$
$d_{BC} = v_3 + v_4$

Propriétés d'ultramétricité :
$v_3 = v_4$
$v_1 = v_2 + v_3 = v2 + v_4$

FIG. 2.10 – *a) un arbre construit à partir d'une distance additive pour quatre espèces. b) un arbre reconstruit à partir d'une distance ultramétrique pour trois espèces.*

2.2.2 Distance entre séquences alignées

Les distances que nous envisagons sont toutes basées sur un alignement préalable des séquences. D'après Swofford et *al.* , il se trouve que l'alignement est la partie la plus difficile et la moins bien contrôlée d'une analyse phylogénétique [SOWH96]. Dans la suite, nous admettrons que cet alignement est effectué de manière correcte.

L'obtention d'un indice de similarité à partir d'un alignement de deux séquences[3] peut être simplement obtenu par la mesure du **taux de similarité**, c'est-à-dire le nombre de sites où l'on retrouve une correspondance entre deux résidus divisé par le nombre de sites total. Une mesure de dissimilarité peut être obtenue en mesurant cette fois le rapport entre le nombre de sites d'erreur et le nombre total de sites. Il se trouve qu'il peut être souvent intéressant de mesurer la similarité entre séquences par une mesure plus fine que la part des sites identiques, en prenant en compte des données supplémentaires telles que, par exemple, les classes auxquelles appartiennent les résidus. Il a donc été introduit des modèles permettant de corriger les distances "brutes" issues de l'alignement.

Définissons F_{XY} comme la matrice de divergence basée sur l'alignement de deux séquences représentatives des espèces X et Y. Pour le cas des séquences nucléiques, cette matrice contient les fréquences relatives de chaque paire de nucléotides:

$$F_{XY} = \begin{pmatrix} \frac{n_{AA}}{N} & \frac{n_{AC}}{N} & \frac{n_{AG}}{N} & \frac{n_{AT}}{N} \\ \frac{n_{CA}}{N} & \frac{n_{CC}}{N} & \frac{n_{CG}}{N} & \frac{n_{CT}}{N} \\ \frac{n_{GA}}{N} & \frac{n_{GC}}{N} & \frac{n_{GG}}{N} & \frac{n_{GT}}{N} \\ \frac{n_{TA}}{N} & \frac{n_{TC}}{N} & \frac{n_{TG}}{N} & \frac{n_{TT}}{N} \end{pmatrix}$$

où n_{ij} représente le nombre de fois où une base i de la séquence X est alignée avec une base j de la séquence Y. N est la somme des n_{ij}.

La distance non corrigée (celle décrite ci-dessus) s'écrit ainsi :

$$p\text{-distance}: \quad \begin{aligned} d_{xy} &= \tfrac{1}{N} \textstyle\sum_{i,j,i\neq j} n_{ij} \\ &= 1 - \textstyle\sum_i \frac{n_{ii}}{N} \end{aligned}$$

Le modèle de Jukes et Cantor [JC69] est basé sur un modèle où il y a équiprobabilité d'apparition de chaque base et le taux de substitution est le même pour toutes les paires de bases:

$$\text{Jukes Cantor}: \quad \begin{aligned} D &= 1 - \textstyle\sum_i \frac{n_{ii}}{N} \\ d_{xy} &= -\tfrac{3}{4} \ln\left(1 - \tfrac{4}{3} D\right) \end{aligned}$$

Notons que la dissimilarité D n'est pas censée dépasser 0.75. Dans le cas contraire, la distance d_{xy} devient indéfinie.

Felsenstein [Fel81] propose un modèle généralisant le modèle de Jukes et Cantor où la fréquence d'apparition des bases n'est plus supposée équiprobable:

$$\text{Felsenstein, 81}: \quad \begin{aligned} D &= 1 - \textstyle\sum_i \frac{n_{ii}}{N} \\ B &= 1 - (\pi_A^2 + \pi_C^2 + \pi_G^2 + \pi_T^2) \\ d_{xy} &= -B \ln\left(1 - \frac{D}{B}\right) \end{aligned}$$

3. Après que l'alignement multiple a été effectué, on prend les couples de séquences et on observe leur alignement comme si il avait été réalisé pour elles deux.

Il apparaît immédiatement que l'on retrouve la distance JC pour $B = 3/4$. Cette distance peut également être utilisée dans le cadre de la comparaison de protéines en fixant par exemple $B = 19/20$.

Kimura [Kim80] propose un modèle à deux paramètres jouant sur la différence entre transversions (Q) et transitions (P):

$$\text{Kimura, 2 paramètres :} \quad \begin{aligned} P &= c+h+i+n \\ Q &= b+d+e+g+j+l+m+o \\ d_{xy} &= \tfrac{1}{2}\ln\left(\tfrac{1}{1-2P-Q}\right) + \tfrac{1}{4}\ln\left(\tfrac{1}{1-2Q}\right) \end{aligned}$$

Felsenstein [Fel84] propose également un modèle à deux paramètres jouant cette fois sur le type de substitutions, distinguant les substituions générales (tout type de substitution) des substitutions à l'intérieur des groupes générant uniquement des transitions:

$$\text{Felsenstein, 84 :} \quad \begin{aligned} \pi_Y &= \pi_C + \pi_T \\ \pi_R &= \pi_A + \pi_G \\ A &= \tfrac{\pi_C\pi_T}{\pi_Y} + \tfrac{\pi_A\pi_G}{\pi_R} \\ B &= \pi_C\pi_T + \pi_A\pi_G \\ C &= \pi_R\pi_Y \\ d_{xy} &= -2A\ln\left(1 - \tfrac{P}{2A} - \tfrac{(A-B)Q}{2AC}\right) + 2(A-B-C)\ln\left(1-\tfrac{Q}{2C}\right) \end{aligned}$$

Une distance basée sur le modèle le plus général, le *most general time-reversible model* ou GTR, à 16 paramètres, a été établie par Lanave et *al.* [LPSS84] et Rodríguez et al. [ROMM90]. Cette distance généralise l'ensemble des distances présentées ci-avant qui n'en sont que des applications particulières.

$$\text{GTR :} \qquad d_{xy} = -\mathrm{tr}\left[\Pi \ln\left(\Pi^{-1}F_{XY}\right)\right]$$

où

$$\Pi = \begin{pmatrix} \pi_A & 0 & 0 & 0 \\ 0 & \pi_C & 0 & 0 \\ 0 & 0 & \pi_G & 0 \\ 0 & 0 & 0 & \pi_T \end{pmatrix}$$

et $\mathrm{tr}(M)$ est la trace de la matrice M.

Chapitre 3

Distances, théorie algorithmique de l'information et compression

Dans notre vie quotidienne, nous faisons souvent appel à la notion de ressemblance. Que ce soit pour identifier l'équivalence entre situations, la nature d'un objet ou même reconnaître notre entourage, sans cesse, nous faisons appel à notre capacité d'analyse comparative.

Si cette faculté semble tout à fait naturelle, il n'est pas aisé de construire des outils permettant la même analyse que celle de l'homme. La difficulté vient en grande partie du fait que les objets que nous avons à comparer sont souvent semblables mais rarement identiques. Ces petites modifications font justement que nous allons juger deux objets identiques ou non. Définir un critère de ressemblance pose le problème de déterminer une limite, un degré de finesse, à partir duquel nous jugerons que les modifications sont significatives ou ne le sont pas.

Ce chapitre est consacré à l'exploration d'une mesure de ressemblance qui se veut universelle : la distance informationnelle. Cette mesure permet de dire si deux objets sont identiques en évaluant la difficulté, mesurée par la quantité d'information à ajouter, qu'il y a de passer de l'un à l'autre. Malheureusement, si cette mesure est satisfaisante d'un point de vue théorique, elle est incalculable. Cependant, il est possible d'en avoir une approximation grâce à la mise au point d'algorithmes de compression. Nous montrons que lorsque nous tentons de comprimer un objet, c'est-à-dire de le décrire d'une manière différente, en se référant à des parties d'un autre objet, nous obtenons une mesure de ressemblance entre ces objets. Cette méthode reste une approximation car il n'est pas possible de définir de manière absolue la meilleure méthode de compression permettant de comprimer un objet à partir d'un autre. Nous ne pouvons en effet explorer l'ensemble des façons possibles de décrire un objet.

Nous commençons par un bref aperçu de la notion de ressemblance spatiale entre objets et de la ressemblance entre textes. Nous abordons ensuite les principes fondamentaux de la théorie algorithmique de l'information. Cela nous permettra de bien comprendre les idées sur lesquelles sont basées les distances entre séquences que nous définissons ici. Après une introduction aux méthodes de compression de données, nous mettons en relation les théories de l'information et ces méthodes. Nous nous attachons ensuite à décrire quelques travaux où une approche par la compression a été utilisée pour l'analyse et la comparaison de séquences génétiques.

3.1 Mesurer la ressemblance

Lorsque l'on parle de comparaison d'objets, la première idée qui vient à l'esprit est la comparaison de la forme et des attributs des objets. La comparaison d'images représente d'ailleurs le plus grand champ d'investigations pour la mise au point de méthodes détectant les ressemblances. C'est pourquoi nous débutons par un bref aperçu de la ressemblance traitée géométriquement et topologiquement.

3.1.1 Approche géométrique et topologique

La ressemblance géométrique se mesure grâce à l'application de transformations. Deux objets vont se ressembler si en effectuant des séries de translations, rotations, symétries et homothéties nous aboutissons à superposer ces objets. Cette notion de ressemblance est extrêmement rigide. En effet, avec un tel concept il est tout à fait impossible de distinguer les différentes parties d'un objet. En particulier, la ressemblance géométrique ne permet pas de dire si deux photos sont proches. Elle sera par contre capable d'identifier la ressemblance entre des volumes, par exemple un cube et parallélépipède rectangle.

Une notion plus souple a été proposée : l'homéomorphie. Cette fois on analyse la topologie de l'objet. En imaginant que chaque surface des objets est molle, on dira qu'ils se ressemblent s'il est possible de déformer la surface de l'un pour la rendre identique à la surface de l'autre (sans découpage de la surface bien sûr). Cette fois la notion de ressemblance est trop souple car elle permet d'identifier comme identiques des objets aussi dissemblables qu'un cube et une sphère.

La notion de ressemblance, on le voit bien ici, est difficile à capter. Il est en tout cas manifestement difficile de dire si oui ou non deux objets se ressemblent. L'introduction d'une mesure de ressemblance s'avère nécessaire. On cherche donc à quantifier la ressemblance et à donner une réponse graduelle à la question de la ressemblance.

Distance de Hausdorff. Elle fournit une bonne approximation de la notion de ressemblance pour des objets représentés par des ensembles de points. La distance de Hausdorff est déterminée de la manière suivante : on superpose les deux ensembles de points de deux objets A et B, on remplace chaque point de A par un disque de rayon suffisamment grand pour que tous les points de B soient recouverts, la distance est le plus petit rayon permettant de recouvrir à la fois A par B et B par A. Pour se donner une image, cela revient à imaginer nos deux objets suffisamment "gonflés" pour qu'ils se recouvrent l'un l'autre. Un exemple est donné figure 3.1.

La distance de Hausdorff est capable d'identifier comme proches de nombreuses figures, l'une en traits pleins, l'autre en traits pointillés; l'une dessinée nettement, l'autre floue. De manière générale les objets dont la forme globale est conservée seront considérés comme semblables. Par contre un carré et un cercle seront considérés éloignés par cette distance. Un mauvais point est que la même figure représentée en inverse (le blanc devient noir et le noir devient blanc) ne sera pas mesurée proche de la figure initiale. De même les réductions ne sont pas correctement détectées.

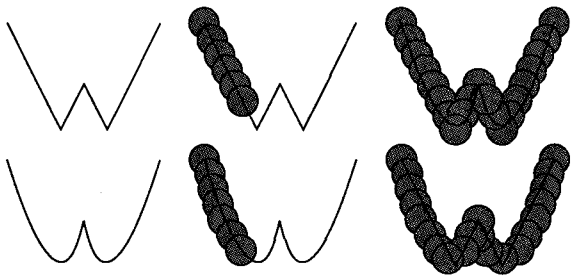


FIG. 3.1 – *Exemple de processus de mesure de la ressemblance avec la distance de Hausdorff. On a remplacé chaque point par un cercle de rayon r dans les deux figures, suffisamment grand pour que la figure du haut puissent s'inscrire dans la figure du bas transformée et inversement.*

3.1.2 Ressemblance entre textes

Précisons immédiatement ce que nous entendons par ressemblance entre deux textes. Il s'agit d'analyser si deux textes se ressemblent, c'est-à-dire s'il existe des sous-parties de l'un dans l'autre et inversement. Nous n'envisageons pas ici les analyses de texte basées sur l'identification d'une sémantique similaire. Nous ne cherchons donc pas ici à savoir si deux textes ont le même sens, mais à savoir si deux textes partagent des sous-parties. Remarquons tout de même que la limite est difficile à fixer et qu'une interprétation de sens peut toujours être associée à la détection d'une similitude, quelle qu'elle soit.

Si l'on s'intéresse aux outils permettant la comparaison de textes, on s'aperçoit qu'il en existe peu. Dans la communauté informatique, le plus connu d'entre-eux est la commande `diff` d'UNIX. Si on donne en entrée deux textes à cette commande, celle-ci indique en sortie les différences qui existent entre ces textes. Ces différences sont les parties des textes qui ne concordent pas si j'imagine mes textes représentés sur des rubans de papiers mis côte à côte. L'algorithme implémenté dans la commande `diff` est un algorithme de recherche de la plus longue sous-séquence commune [Mye86, CR94]. Ce problème a pour but de trouver un sous ensemble de segments des deux textes et qui respectent l'ordre naturel des textes (leur sens de lecture).

En fait, il existe une abondante littérature sur les algorithmes permettant d'identifier des propriétés sur les textes : recherche de toutes les répétitions, recherche de la plus longue sous-séquence commune, recherche du plus grand segment commun [CR94]. La mise en oeuvre de ces outils dans le cadre de la comparaison globale de textes aboutit finalement à une analyse proche de l'alignement de séquences génétiques.

Ainsi, on n'oubliera pas de citer les distances de **Hamming** et de **Levenshtein** qui mesurent le nombre minimal d'opérations unitaires nécessaires pour transformer un texte en un autre. Les opérations sont la substitution, l'insertion et la délétion dans le cas de la distance de Levenshtein, et la substitution seule dans le cas de la distance de Hamming.

3.2 Notions de théorie algorithmique de l'information

Nous avons vu dans le chapitre précédent comment était abordée la notion de mesure de ressemblance entre séquences génétiques. Nous décrivons maintenant un autre point de vue qui constitue le fondement des idées qui ont été développées au cours de ce travail.

3.2.1 Introduction

Les données que nous manipulons nécessitent toujours d'être décrites. Cette description est le plus souvent très simple et consiste uniquement en l'énumération de la dite donnée. La manière la plus simple de décrire le texte "ATATAT" est de l'écrire explicitement, c'est-à-dire écrire les lettres les unes après les autres.

Pour des raisons de gain en espace ou de gain en coût de transmission au travers de réseaux, il arrive que la description d'un objet soit écrite de manière plus compacte. On parle alors de compression de l'objet (voir section 3.3). Par exemple, on peut écrire "3×AT" à la place de "ATATAT".

Un algorithme de compression calcule donc une nouvelle description d'un objet en essayant de faire que cette description soit plus courte. Pour diminuer la taille de la description de l'objet, l'algorithme de compression (ou compresseur) doit utiliser au maximum les régularités que comporte l'objet.

Ainsi, compresser un objet peut avoir un tout autre but que le gain en espace. En effet, l'exploitation des régularités de l'objet pour en obtenir une description plus courte peut servir à l'analyse de celui-ci. Nous reviendrons plus tard sur cette interprétation. Nous présentons maintenant les fondements théoriques de cette idée.

Nous dirons qu'une méthode de compression est optimale si elle parvient à utiliser toutes les régularités de l'objet qu'elle compresse. La description optimale d'un objet exhibe donc la partie de cet objet qui ne peut être comprimée et par là même définit la **quantité d'information** contenue dans l'objet. Elle est également la description la plus courte. Cette description la plus courte correspond à la notion de programme le plus court défini par la **complexité de Kolmogorov**. Après avoir défini cette notion, nous nous intéresserons au principe de description de longueur minimale puis à la notion de distance informationnelle.

Machine de Turing. La définition de la complexité de Kolmogorov est basée sur la mesure de la longueur de programmes d'une machine de Turing universelle. Nous introduisons brièvement ce modèle abstrait de machine.

Une machine de Turing est composée d'une bande infinie subdivisée en cellules et d'une tête capable d'effectuer les actions suivantes : lire et écrire dans les cellules, se déplacer à gauche ou à droite sur la bande. Chaque cellule peut contenir soit '0', soit '1', soit un blanc que nous notons 'B'. Une représentation est donnée figure 3.2. Il existe également un ensemble d'états dans lequel peut se trouver la tête. Au départ, la bande ne contient que des 'B' excepté un certain nombre fini de cellules adjacentes qui contiennent la donnée, la tête est positionnée sur la première cellule ne contenant pas de 'B' et se trouve dans l'état de départ. Un ensemble de règles de la forme (p,s,a,q) où p est l'état courant, s le symbole attendu à la lecture, a l'action à effectuer et q l'état dans lequel se trouve la tête après exécution de la règle, régissent le déplacement de la tête. Une fois que la tête se trouve dans un état final, l'exécution est terminée et la lecture de la bande fourni le résultat.

Exemple 3.1 *Nous donnons l'exemple d'une machine de Turing permettant de multiplier un nombre par 2. Un tel calcul se fait simplement à partir de l'écriture d'un nombre en binaire en ajoutant un '0' à la fin du nombre :* $6 = 110$ *devient* $1100 = 12$.

La machine possède deux états : q_0 *est l'état initial et* q_1 *est l'état final. L'ensemble des règles*

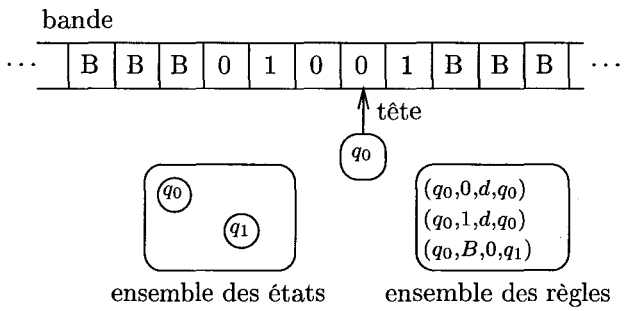


FIG. 3.2 – *Machine de Turing*

est le suivant :

$$\{(q_0,1,d,q_0),(q_0,0,d,q_0),(q_0,B,0,q_1)\}$$

La première règle (resp. la seconde) signifie que si dans l'état q_0 on lit un '1' (resp. un '0') alors la tête se déplace vers la droite et on reste dans l'état q_0, la troisième que si dans l'état q_0 on lit un 'B' alors on écrit un '0' et on passe dans l'état q_1.

Le schéma ci-dessous représente l'évolution de la bande :

Étape	*État*	*Règle*	*Bande*
départ	q_0		... BB[1]10BB ...
1	q_0	*1*	... BB1[1]0BB ...
2	q_0	*1*	... BB11[0]BB ...
3	q_0	*2*	... BB110[B]B ...
4	q_0	*3*	... BB110[0]B ...
fin	q_1		... BB110[0]B ...

A la fin, la bande contient le résultat.

Une **machine de Turing universelle** est capable de simuler le fonctionnement de n'importe quelle autre machine de Turing.

L'ensemble des fonctions $f : \{0,1\}^* \to \{0,1\}^* \cup \{0,1\}^\infty$ est appelé l'ensemble des **fonctions partiellement récursives** : pour toute fonction récursive partielle, il existe une machine de Turing calculant cette fonction. Comme une machine de Turing universelle peut imiter toute machine de Turing, une machine de Turing universelle est capable de calculer l'ensemble des fonctions partiellement récursives.

3.2.2 De Shannon à Bennett en passant par Kolmogorov

Nous évoquons ici trois théories de l'information dans l'ordre chronologique de leur apparition. Chacune propose une mesure qui se veut objective pour la quantité d'information contenue dans un objet. Seule la complexité de Kolmogorov, qui est la notion centrale de la théorie algorithmique de l'information, interviendra dans la suite. Cependant il nous a semblé intéressant de voir de quelle manière la mesure du contenu en information pouvait être envisagée.

Dans la suite nous supposerons qu'un objet peut toujours être représenté sous forme d'un texte. Nous confondrons donc les termes chaîne, texte et objet.

La théorie de la transmission de Shannon

La **théorie de la transmission** de Shannon [SW49] permet de mesurer la **valeur en information** d'une chaîne de m caractères que l'on veut transmettre à un récepteur lorsque l'on dispose de connaissances sur la fréquence des lettres de l'alphabet qui composent m. Considérons par exemple l'ensemble des chaînes binaires de longueur 9999, la transmission d'une de ces chaînes nécessite en moyenne 9999 bits.

Cette théorie est donc basée sur un modèle probabiliste de distribution des lettres de l'alphabet sur lequel est écrit le message. Cela implique que le récepteur possède une certaine connaissance et un certain pouvoir de calcul.

Si l'on veut coder le message composé de 9999 '1', il est possible de procéder de la manière suivante : il faut 14 bits pour exprimer 9999 en binaire, puis il faut préciser le message répéter 9999 fois, '1'. Cet exemple montre les limites de la théorie de Shannon : elle exprime la quantité moyenne d'information d'une famille de messages et non le contenu en information d'un message précis. Elle doit donc être vue comme une théorie probabiliste de l'information [Del94].

La complexité de Kolmogorov

Une seconde notion de théorie de l'information a été introduite par Kolmogorov [Kol65] et s'énonce comme suit:

> *La **complexité de Kolmogorov** (ou entropie algorithmique) d'un objet fini x est la longueur du plus court programme binaire permettant le calcul de x sur une machine de Turing universelle.*

La complexité de Kolmogorov est parfois appelée complexité de Chaitin-Kolmogorov.

Définition 3.1 *La **complexité de Kolmogorov** $K_U(x)$ d'une chaîne par rapport à une machine universelle U est définie par :*

$$K_U(x) = \min\{l(p) \ : \ U(p) = x\}$$

où $l(p)$ désigne la longueur du programme p et $U(p)$ le résultat de l'exécution du programme p sur la machine U.

La notion de programme employée ici est la même que la notion de description employée dans l'introduction de ce chapitre. $K(x)$ représente la longueur de la meilleure description que l'on puisse obtenir de l'objet x. C'est la longueur de la description obtenue par le passage de x au travers d'un compresseur optimal pour x.

La complexité de Kolmogorov est indépendante (à une constante additive près) de la machine sur laquelle le programme minimal est exécuté. Elle représente une mesure absolue et objective de la quantité en information contenue dans un objet.

Théorème 3.1 (d'invariance) *Si U est une machine universelle, alors pour tout autre machine universelle A,*

$$K_U(x) \leq K_A(x) + c_A$$

pour toutes chaînes $x \in \{0,1\}^*$, où la constante c_A ne dépend pas de x.

Preuve 3.1 ([CT91])

Supposons que nous ayons un programme p_A pour la machine A imprimant x : $A(p_A) = x$. Nous pouvons faire précéder ce programme d'un programme de simulation s_A indiquant à U comment simuler la machine A. La machine U interprétera les instructions du programme comme la machine A et imprimera x. Le programme imprimant x pour U est $p = s_A p_A$ et sa longueur est :

$$l(p) = l(s_A) + l(p_A) = c_A + l(p_A)$$

où c_A est la longueur du programme de simulation. Ainsi,

$$K_U(x) = \min_{p:U(p)=x} l(p) \leq \min_{p:A(p)=x} (l(p) + c_A) = K_A(x) + c_A$$

pour toute chaîne x. □

Intuitivement, $K(x)$ représente la quantité minimale d'information nécessaire à la génération de x et la complexité de Kolmogorov d'un objet x ne peut pas excéder la longueur de cet objet à une constante près.

Théorème 3.2 *Soit U une machine de Turing universelle. Il existe une constante c telle que pour tout x :*

$$K(x) \leq l(x) + c_U$$

Le programme le plus long de référence est noté `print x` où `x` est donné explicitement. Il comprend donc l'écriture binaire de `x`.

Il est également possible de définir la complexité d'un objet x relativement à un objet y. Cela définit la complexité de Kolmogorov relative entre deux objets:

*La **complexité de Kolmogorov relative** (ou conditionnelle) d'un objet fini x relativement à un objet y est la longueur du plus court programme binaire permettant le calcul de x sur une machine universelle si y est fourni comme donnée auxiliaire. On note $K(x|y)$ cette longueur.*

Définition 3.2 *La **complexité de Kolmogorov relative** $K_U(x|y)$ d'une chaîne par rapport à une machine universelle U est définie par :*

$$K_U(x|y) = \min\{l(p) \ : \ U(p,y) = x\}$$

$K(x|y)$ représente la quantité d'information nécessaire à la génération de x, y étant donné. Elle représente donc la quantité minimale d'information qu'il faut ajouter à y pour trouver x. Il est possible d'énoncer un théorème similaire au précédent montrant que la complexité de Kolmogorov relative d'un objet x ne peut jamais excéder la complexité de Kolmogorov de l'objet x lui-même.

Théorème 3.3 *Il existe une constante c telle que pour tout x et y :*

$$K(x|y) \leq K(x) + c$$

Malheureusement, il est prouvé que la complexité de Kolmogorov est incalculable. L'indécidabilité de l'arrêt d'un programme implique que nous ne pouvons envisager l'ensemble des programmes capables d'engendrer une donnée d. Puisque nous ne savons pas si ces programmes s'arrêtent, on ne peut pas tous les essayer. Nous ne pouvons donc pas avoir de certitude quant à la minimalité de la longueur d'un programme engendrant une donnée d.

Théorème 3.4 *La fonction qui à une chaîne s associe $K(s)$ n'est pas récursive.*

Cela ne signifie pas pour autant qu'il est impossible d'utiliser la complexité de Kolmogorov. En pratique, nous pouvons calculer une approximation de $K(x)$, par valeurs supérieures. Il n'est cependant pas possible d'évaluer la qualité d'une telle approximation.

La notion abordée ici est malgré tout plus intéressante que la théorie de la transmission de Shannon car elle est une mesure absolue de la quantité d'information d'un objet individuel contrairement à une mesure en moyenne de la quantité d'information nécessaire à la transmission d'un objet.

En particulier, la notion de complexité relative introduite ici est particulièrement intéressante pour effectuer la comparaison d'objets. Cette quantité représente la longueur minimale d'un programme produisant une donnée x à partir d'une donnée y. Imaginons que y et x ne soient pas semblables, alors $K(x|y)$ sera proche de $K(x)$ puisque le programme le plus court ne tirera aucun profit de la donnée de y. En revanche, si x et y sont semblables, alors des parties de x pourront être décrites grâce à des parties de y. Cela peut réduire significativement la quantité d'information à donner pour décrire x. Le cas le plus favorable est celui où x et y sont identiques. Il suffit alors de dire que x est y, qui, on l'imagine bien, est une description très courte de x (mais pas de longueur nulle) par rapport à la description complète de x.

La profondeur logique de Bennett

Bennett [Ben88] introduit la **profondeur logique** d'un objet x comme le temps de calcul nécessaire à la production de x sur une machine de Turing universelle à partir de la description minimale de x (au sens de la complexité de Kolmogorov).

La valeur en information de x est donc cette fois le nombre minimum de pas de calcul nécessaires à la production de x à partir du plus court programme générant x. Bennett montre que cette notion est robuste : elle dépend peu de la machine sur laquelle on exécute le programme minimal.

Cette notion est intéressante car elle permet la mesure de "l'historique" d'un objet en ce sens qu'elle mesure toute l'information engendrée pour la création de cette objet. Par exemple, la suite des digits de π (le premier million par exemple) possède une complexité de Kolmogorov petite car il existe des algorithmes très courts permettant de les engendrer. Au contraire, la profondeur de Bennett sera assez grande car pour engendrer ces digits il est nécessaire d'exécuter un grand nombre de calculs [Del94].

3.2.3 Principe de Description de Longueur Minimale (MDL principle)

Hormis la complexité de Kolmogorov que nous avons décrite précédemment, une autre notion importante de la théorie algorithmique de l'information est la loi de distribution *a priori* des

programmes appelée **distribution de Solomonov-Levin**.

Cette loi formalise le **principe du rasoir d'Occam** qui affirme que

"Les explications les plus courtes sont les plus probables"

La distribution de Solomonov-Levin considère donc que les programmes (ou les descriptions) les plus courtes sont les plus probables. Ce principe s'appuie sur le fait qu'*a priori*, sans hypothèse supplémentaire, tous les programmes de longueur infinie d'une machine de Turing sont équiprobables. Il existe une application capable de transformer une séquence infinie (construite avec des '0' et des '1') en un réel compris dans l'intervalle [0,1]. La distribution des séquences infinies est donc la même que la distribution uniforme sur [0,1].

Les séquences infinies peuvent être partitionnées en deux catégories : celles qui débutent par '0' et celles qui débutent par '1'. Il est possible de repartionner l'ensemble des séquences débutant par '0' en deux catégories : celles débutant par '00' et celles débutant par '01'. On imagine bien la définition d'une application bijective qui à toute séquence finie de '0' et de '1' fait correspondre un sous-intervalle de [0,1]. En partant de l'intervalle [0,1], on construit par itérations sur les lettres de la séquence l'intervalle représentant celle-ci en appliquant les deux règles suivantes : si on lit un '0' on considère pour la suite la moitié inférieure de l'intervalle (par exemple, on passe de [0,1] à [0,1/2]), et si on lit un '1', on considère l'autre moitié (par exemple, on passe de [0,1] à [1/2,1]). L'intervalle correspondant à '011' sera construit de la manière suivante :

$$[0,1] \xrightarrow{0} [0,1/2] \xrightarrow{1} [1/4,1/2] \xrightarrow{1} [3/8,1/2]$$

Ainsi on dispose d'une application qui a toute séquence finie de '0' et de '1' fait correspondre l'ensemble des séquences infinie l'ayant pour début et donc le sous intervalle de [0,1] lui correspondant. Si l'on considére que tous les programmes d'une longueur infinie sont équiprobables, la largeur de l'intervalle auquel est associé un programme de longueur fini est proportionnel à sa probabilité et donc les programmes les plus courts sont les plus probables.

Le **principe de description de longueur minimale** s'énonce ainsi [LV93] :

La meilleure théorie pour expliciter un ensemble de données est celle qui minimise la somme de :

- *la longueur, en bits, de la description de la théorie; et*
- *la longueur, en bits, de la donnée lorsqu'elle est codée par cette théorie.*

Pratiquement, cela implique que sans connaissance particulière d'un objet, on doit préférer un modèle dont la description est la plus courte possible pour expliquer la manière dont l'objet est obtenu. Le modèle dont il est question ici est sous-jacent au programme de longueur minimal. Si on dispose de ce programme, on connaît alors les propriétés qui ont permis d'obtenir une description plus courte que la donnée directe de l'objet. Ces propriétés définissent le modèle qui explique l'objet. Si nous reprenons l'exemple du texte `ATATAT`, et si nous supposons que le programme le plus court est effectivement "3×`AT`", alors la propriété qui permet d'écrire ce programme est la répétition trois fois du motif `AT`. Le modèle sous lequel la séquence est décrite est donc la répétition successive de `AT`.

Ce point de vue est à rapprocher du principe de parcimonie plus commun dans le cadre biologique. Il signifie que l'on va toujours préférer une explication minimisant le nombre d'événements

à une autre. Ce principe est directement appliqué lors de la reconstruction d'arbres phylogénétiques en minimisant le nombre d'événements évolutifs. Il guide également la recherche d'un meilleur alignement par minimisation d'une fonction de score et par conséquent minimisation du nombre de mutations nécessaires entre deux séquences.

3.2.4 Distance informationnelle [BGL+98]

Bennett, Gács, Li, Vitányi et Zurek étudient dans [BGL+98] une mesure absolue de l'information entre deux objets. Cette mesure est basée sur la complexité relative de Kolmogorov.

Remarquons dans un premier temps que l'information qu'il faut ajouter à un objet x pour obtenir un objet y n'est pas forcément la même que celle nécessaire pour obtenir x à partir de y. Par exemple $K(\varepsilon|x)$ est petit quelque soit x. Par contre $K(x|\varepsilon)$ sera égal à $K(x)$ et dépend donc de x et de sa taille. L'utilisation directe de $K(x|y)$ ne permet pas de fournir une distance à cause de son asymétrie.

Supposons que l'inégalité suivante soit vérifiée pour deux objets x et y :

$$K(x|y) < K(y|x)$$

Une façon de résoudre le problème est de définir la distance comme la somme des deux quantités, $K(x|y)+K(y|x)$. Le problème est qu'alors on surestime la quantité d'information nécessaire pour passer de x à y et de y à x si il existe une information commune au passage de l'un à l'autre.

Il est possible de définir une distance mesurant toute l'information nécessaire à l'obtention de x à partir de y et à l'obtention de y à partir de x.

Définition 3.3 *La **distance max** E_1 entre x et y est définie par :*

$$E_1(x,y) := \max\{K(x|y),K(y|x)\}$$

Posons $k_1 = K(x|y)$, $k_2 = K(y|x)$ et $l = k_2 - k_1$. On suppose que $k_1 \leq k_2$. Alors il est possible de montrer [LV93] qu'il existe deux chaînes, q de longueur $l + O(\log l)$ et d de longueur $k_1 + O(\log k_1)$ telles que d est un programme minimal capable de calculer à la fois xq à partir de y et y à partir de xq. Cela montre que $E_1(x,y)$ est égal à la longueur du plus court programme $p := qd$ permettant de calculer x à partir de y et y à partir de x.

De manière plus explicite, le passage de x à y se déroule ainsi : ajouter q à x, utiliser ensuite d pour passer de qx à y. Pour passer de y à x : utiliser d pour passer de y à qx puis supprimer q. Le passage de x à y utilise donc le programme $p = qd$ alors que le passage de y à x utilise le programme q.

Une autre distance peut être définie en utilisant des machines de Turing réversibles. De manière très succincte, une machine de Turing réversible permet de retrouver la donnée initiale à partir du résultat du calcul, ceci est possible si l'on dispose d'une bande qui permet de stocker l'historiques des calculs effectués [Lec63].

Il est possible de définir la complexité de Kolmogorov relative sur des machines de Turing réversibles de manière similaire à la définition de la complexité de Kolmogorov relative sur des

machines classiques. Une machine de Turing réversible est notée UR et la compléxité de Kolmogorov définie sur ces machines KR.

Définition 3.4 *La distance réversible* $E_2(x,y)$ *entre* x *et* y *est définie par :*

$$E_2(x,y) := KR(y|x) = \min\{l(p) \ : \ UR(p,x) = y\}$$

Les auteurs de [BGL+98] démontrent d'une part que E_2 vérifie l'inégalité triangulaire à une constante additive près, et d'autre part que E_1 est égal à E_2 à une constante logarithmique près.

Théorème 3.5

$$E_2(x,z) < E_2(x,y) + E_2(y,z) + O(1)$$

Théorème 3.6

$$E_1(x,y) = E_2(x,y) + O(\log)$$

Plus généralement, nous dirons que la **distance informationnelle** est la longueur du plus court programme permettant à la fois de produire x en ayant y comme argument et inversement. La distance informationnelle est une mesure de la similarité entre des objets x et y puisqu'elle mesure l'information nécessaire au passage entre x et y.

La section suivante s'intéresse aux méthodes de compression de données et plus particulièrement aux méthodes de compression relative permettant de comprimer un objet en utilisant les parties d'un autre. Nous allons voir comment ces méthodes de compression permettent un calcul approché de la distance informationnelle. Les fondements théoriques de la mesure que nous définissons dans ce travail sont ceux de la distance informationnelle. La mise en œuvre est basée sur l'application d'une méthode de compression relative au calcul de la similarité entre séquences basée sur un modèle spécifique.

3.3 Compression et analyse de séquences

Nous avons évoqué à la section précédente l'idée de compression de manière très succincte. Nous développons plus avant la notion de compression, de compresseur et de séquence compressible dans le cadre de cette section. Nous terminons par un aperçu des travaux qui ont été effectués sur l'utilisation de méthodes de compression pour l'analyse de séquences génétiques après avoir discuté du rapport très étroit qui existe entre les méthodes de compression et les théories de l'information.

3.3.1 Notion de compresseur

Un compresseur peut être défini comme un mécanisme de codage permettant de réécrire un texte sous une forme plus courte que sa forme initiale. Un compresseur tente donc d'exploiter les propriétés de redondance d'un texte afin de les éliminer et donc de réduire la taille du texte. Cette réduction peut bien évidemment servir à réduire l'espace occupé par un texte sur un périphérique de stockage ou au travers d'une voix de communication. Mais le fait qu'un compresseur mette en avant les propriétés d'un texte permet aussi d'utiliser la compression comme moyen d'analyse. Ainsi, si un compresseur est basé sur un certain modèle, l'application de ce compresseur à un

texte permet d'évaluer la concordance entre le modèle et le texte. En effet, plus le modèle est pertinent pour un texte, plus ce texte sera comprimé.

Mesure du gain

L'évaluation de la pertinence d'un modèle nécessite de disposer d'une mesure. Pour cela, on utilise les mesures du **gain** et du **taux de compression**.

Pour comparer des choses comparables, il faut que les textes soient écrits dans le même alphabet. Nous supposerons que les textes que nous manipulons sont toujours écrits en binaire (soit sur l'alphabet {0,1}). La longueur d'un texte sera alors simplement le nombre de bits qui le compose. Par exemple, chaque base de l'ADN, il en existe quatre, peut être codée sur 2 bits : `A`=00, `T`=01, `C`=10 et `G`=11. La longueur du texte `ATTG` en binaire sera donc de 8.

Le gain de compression est simplement le nombre de bits gagnés lors du passage de la version originale du texte à sa version comprimée:

$$\text{gain} = \text{longueur du texte original} - \text{longueur du texte comprimé}$$

Cette mesure est absolue et ne permet pas de comparer l'efficacité d'un compresseur sur différents textes.

Le taux de compression apporte quant à lui une mesure relative, et donc comparable. Il est défini ainsi:

$$\begin{aligned}\text{taux de compression} &= 1 - \frac{\text{longueur du texte original} - \text{longueur du texte comprimé}}{\text{longueur du texte original}} \\ &= \frac{\text{longueur du texte comprimé}}{\text{longueur du texte original}}\end{aligned}$$

Plus le taux de compression est élevé, moins le texte a été comprimé.

Par exemple, si nous disposons d'un texte de longueur 100 bits et que sa version comprimée est de longueur 40 bits, le gain sera de 60 bits et le taux de 0.4.

Avec ou sans perte d'informations

Lorsque l'on utilise un compresseur, notamment lorsqu'il s'agit de stocker ou de transmettre des données, on s'attend à ce qu'il existe une méthode permettant de retrouver le texte initial : un **décompresseur**. On distingue les méthodes **conservatives**, pour lesquelles il existe un décompresseur capable de reconstruire exactement le texte initial, des méthodes **non-conservatives**, pour lesquelles il existe un décompresseur qui ne restitue pas exactement le texte initial..

Les méthodes non-conservatives sont aussi appelées méthodes avec perte d'informations car le texte comprimé ne contient alors plus toute l'information présente dans le texte original. Si toute l'information restait présente, alors il existerait une méthode de décompression capable de fournir le texte original.

L'utilisation d'une méthode non-conservative n'est pas forcément sans intérêt. Pour le stockage de données, il existe des méthodes de compression d'images avec perte d'informations. Pour la compression d'image, les méthodes non-conservatives sont très utilisées. La qualité de l'image

décompressée étant satisfaisante pour l'usage que l'on veut en faire (par exemple un simple affichage à l'écran sans avoir pour objectif une impression de qualité photo). En particulier, le fait de se passer de l'auto-délimitation (voir paragraphe suivant) peut simplifier la mise au point d'un compresseur tout en fournissant un outil d'analyse satisfaisant.

Schéma de codage

L'élaboration d'une méthode de compression nécessite de se définir un **schéma de codage**. Le schéma de codage est un ensemble de règles déterminant la façon dont doivent être codées les parties du texte que nous allons réécrire dans le texte comprimé. **C'est une fonction injective qui applique un mot source sur un mot code** [Sto88]. Si l'on veut que la méthode de compression soit conservative, il faut que le codage ne soit pas ambigu, c'est-à-dire qu'il faut que la fonction de codage soit bijective.

L'élaboration de ce schéma est faite soit en même temps, soit *a posteriori* de la phase d'analyse du texte que l'on veut compresser. La plupart des compresseurs de textes classiques fonctionnent en entremêlant les deux phases : le codage et le texte comprimé sont construits à la lecture du texte original. D'autres méthodes nécessitent d'analyser le texte pour en extraire de l'information et ensuite construisent le codage à partir de cette analyse.

Pour que la méthode de compression révèle bien la propriété que l'on veut exprimer, le codage associé doit être performant, c'est-à-dire que le codage doit minimiser la longueur de la séquence comprimée. Il est bien souvent difficile de prouver qu'un codage est performant, c'est-à-dire qu'il conduit à une description de longueur minimale, et ce sont des arguments de bon sens qui permettent de justifier un codage [RD98].

L'**auto-délimitation** fournit une manière simple d'obtenir un codage non ambigu. Il s'agit de construire un code où **aucun mot code n'est le préfixe d'un autre mot code**.

Outre le fait qu'il soit non ambigu, l'intérêt d'un codage auto-délimité est qu'il assure un décodage en temps linéaire. En effet, lors de la lecture du texte comprimé, on détecte facilement la fin d'un mot code u parce qu'il n'existe pas de mot code ayant pour préfixe u. Il suffit alors d'un passage sur le texte comprimé pour réussir à le décoder.

L'auto-délimitation la plus simple consiste en l'utilisation d'un **format fixe** pour représenter les données. Par exemple, si l'on manipule des entiers dont on sait que leur valeur est contenue dans l'intervalle [0..255], on pourra coder les entiers en utilisant 8 bits, quel que soit l'entier que l'on code. Ainsi le codage de trois entiers nécessitera $3 \times 8 = 24$ bits. On est certain de pouvoir décoder des suites d'entiers codés de cette manière car le code résultat d'un tel codage est toujours multiple de 8. Il suffit alors de prendre le nombre comprimé, de diviser la séquence de bits qui le compose en groupes de 8 bits et de décoder un à un chaque groupe de 8 bits. Un tel codage est particulièrement simple et ne semble pas optimiser la compression en ce sens que le codage du chiffre 1 occupe le même espace que le codage du nombre 255. Cependant il existe bien des cas où la longueur moyenne des codes que l'on définit est proche du format fixe. Son utilisation est alors tout à fait appropriée.

D'autres façons d'auto-délimiter plus sophistiquées existent. L'exemple suivant en illustre une. Admettons cette fois que nous codions les entiers de la façon suivante : si x est un entier écrit en binaire, son code sera $1^{|x|}0x$, c'est-à-dire autant de '1' qu'il faut de bits pour écrire x en binaire, puis '0', puis l'écriture de x en binaire. Le chiffre 5, dont l'écriture binaire est 101

aura pour code 1110101. Il apparaît assez clairement qu'il est facile de retrouver x à partir de son code. Démontrons maintenant que cela fournit un code auto-délimité. Si x et x' ont la même longueur et que x est préfixe de x', alors nécessairement x et x' seront les mêmes. Si x a une écriture binaire plus courte que x' alors on doit avoir $1^{|x|}0$ préfixe de $1^{|x'|}0$, ce qui est impossible. Enfin si x a une écriture binaire plus longue que x' alors le code de x est plus long que le code de x', ce qui est impossible.

Compression relative

Dans le cadre de la compression relative, on dispose en plus du texte à compresser, nommé ici **texte cible**, d'un texte de référence, que nous nommerons **texte source**. L'objectif est alors de comprimer le texte cible en exploitant les régularités qu'il peut exister entre le texte source et le texte cible. L'idée du schéma de codage reste la même.

Le but est donc ici d'exprimer le texte cible de la manière la plus courte possible en s'aidant du texte source. Dans le cadre dans lequel nous envisageons la compression relative, nous n'autorisons pas la méthode de compression à tirer partie à la fois des régularités entre le texte source et le texte cible et des régularités du texte cible lui-même. Seul le texte source doit servir au codage. Cela signifie que les seules parties du texte cible utilisées doivent être données explicitement. Par exemple, si mon texte source est s =`ATCT` et mon texte cible est t =`AATCGAGAGA`, la compression de t par rapport à s ne pourra pas être `A,s[0,3],3×GA` car `3×GA` n'est pas une régularité obtenue à partir de s.

Techniques de codage

Nous proposons maintenant une "classification" des algorithmes de compression suivant le point de vue qu'ils adoptent pour définir leur schéma de codage. Les catégories que nous proposons peuvent être discutées mais notre but n'est pas de définir un classement absolu mais plutôt de faire ressortir les différentes sensibilités inhérentes aux différents types de compresseurs. Nous faisons particulièrement la distinction entre les **compresseurs statistiques** et les **compresseurs algébriques**. Les premiers sont basés sur l'idée qu'un texte est le résultat d'un processus aléatoire, choisissant chaque lettre du texte suivant une loi de probabilité (la régularité peut porter sur des couples de lettres, des triplets, etc.). Le compresseur est basé sur cette loi. Comme elle est la plupart du temps inconnue, le schéma de codage, c'est-à-dire le modèle de l'algorithme de compression tente d'approcher cette loi. Les seconds sont basés sur l'identification de propriétés plus proches de la structure du texte : des répétitions, des déplacements, etc. Il s'agit donc de propriétés locales alors que les propriétés identifiées par les régularités statistiques sont globales.

Compression statistique. La conception d'une méthode de compression peut être basée sur des données statistiques du texte. C'est-à-dire que la méthode va utiliser des informations de nature statistique pour comprimer le texte.

Ces méthodes sont basées sur l'idée que *le modèle de processus d'une séquence de caractères est un processus aléatoire qui tire chacun des symboles au sort dans l'alphabet, en respectant les probabilités d'apparition données.*

Le compresseur va donc chercher à affecter un code court aux symboles qui apparaissent le plus souvent et un code plus long à ceux qui apparaissent le moins souvent.

On peut citer le codage de **Huffman** et le codage **arithmétique**. Le premier est tel que chaque symbole est codé sur un nombre déterminé de bits. Ce nombre dépend de la distribution de probabilité p de chaque lettre de l'alphabet (figure 3.3). Le second code un texte sur un intervalle de nombres réels. L'intervalle [0,1] est divisé en sous-intervalles en suivant la distribution de probabilité p. Suivant la lettre que l'on code dans la séquence, le sous-intervalle correspondant est de nouveau subdivisé en sous-intervalles suivant la loi donnée par p (figure 3.4). Le texte est codé par la suite des codes correspondants à chacun des symboles.

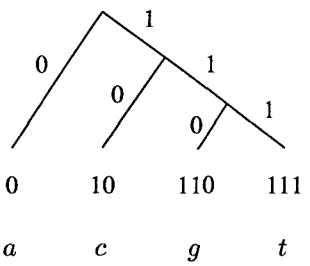


FIG. 3.3 – *Exemple de codage de Huffman.* $p(a) = 1/2$, $p(c) = 1/4$, $p(g) = 1/8$, $p(t) = 1/8$. *'a' a une plus forte probabilité d'apparition que les autres lettres, il est donc codé sur un plus petit nombre de bits.*

La distribution de probabilités des symboles est le **modèle** de l'algorithme de compression. Plus le modèle sera proche de la réalité et plus le texte comprimé sera court, et inversement.

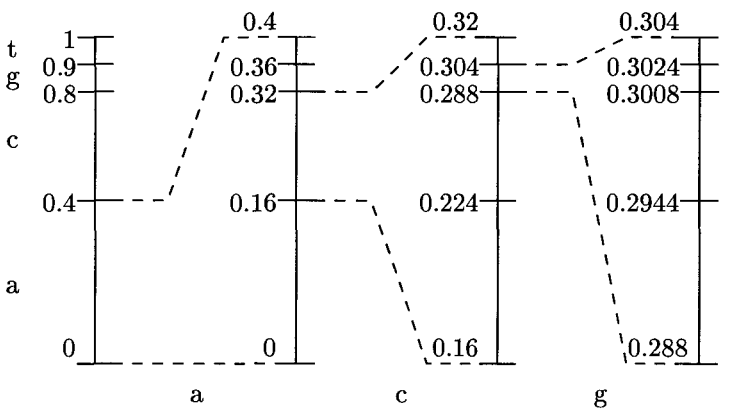


FIG. 3.4 – *Exemple de codage arithmétique.* $p(a) = 2/5$, $p(c) = 2/5$, $p(g) = 1/10$, $p(t) = 1/10$. *Le codage du mot* `a` *est* [0,0.4], *le codage de* `ac` *est* [0.16,0.32] *et le codage de* `acg` *est* [0.288,0.304].

Compression algébrique. Parmi les méthodes algébriques, nous allons faire la distinction entre les méthodes basées sur des références à un dictionnaire et celles basées sur des références directes au texte, que nous appelons méthodes de **compression algorithmiques**. Illustrons la différence sur l'exemple du texte `ATATATATATATATATATAT`. Un compresseur à dictionnaire va disposer dans son dictionnaire du mot `AT` portant un numéro, disons 2. La texte comprimé de `ATATATATATATATATATAT` sera "2222222222", c'est-à-dire la suite de références aux mots du dictionnaire. Dans le cas du compresseur algorithmique, le texte comprimé sera "10×`AT`". L'interprétation de la lecture du texte comprimée est bien différente, d'un côté on a repéré que la

structure du texte était le mot AT, puis le mot AT, puis le mot AT, *etc.* , alors que de l'autre la structure est la répétition 10 fois de AT. Cet exemple est simpliste mais il illustre la différence que nous voulons mettre en avant ici.

Les **compresseurs à dictionnaire** vont donc coder chaque facteur par une référence à ce facteur dans un dictionnaire. On appelle également ce type de compresseur **compresseur substitutionnel** parce que le codage d'un mot du texte est obtenu par substitution de ce mot par une référence. Le dictionnaire est soit construit sur place, c'est-à-dire au fur et à mesure de la lecture, soit est connu à l'avance. Cette technique est adoptée dans les algorithmes de Ziv et Lempel, LZ77 et LZ78 [ZL77, ZL78], qui constituent la base des algorithmes actuels de compression de texte à usage généraliste.

LZ77 fonctionne par l'intermédiaire d'une fenêtre que l'on fait glisser le long du texte de gauche à droite. Pour une position donnée p, on recherche le plus long segment débutant à cette position et présent dans les n caractères précédents. n est la taille de la fenêtre et est fixé par avance. De plus la taille de la répétition est limitée à une longueur maximale t. On remplace ce segment par un pointeur constitué d'un couple (position, longueur) qui indique exactement où se trouve l'occurrence antérieure du segment.

LZ78 est une amélioration où la taille de la fenêtre n'est plus limitée. Cette fois, le dictionnaire est stocké et mis à jour au fur et à mesure de la lecture du texte. La taille du dictionnaire est limitée par la taille (en bits) des numéros permettant de référencer les mots. La question de la gestion du dictionnaire et particulièrement de sa remise à jour une fois qu'il est rempli a donné lieu à de nombreuses variantes. L'avantage de cette méthode par rapport à la précédente est l'accès rapide aux mots du dictionnaire et la non limitation de la fenêtre de recherche. De plus, si le dictionnaire n'est pas plein alors il n'est pas nécessaire de coder les références sur la taille maximale des numéros d'ordre autorisée. En effet, le compresseur et le décompresseur construisant le dictionnaire de la même manière, il est possible de faire varier la taille des références sans perdre d'information.

Les **compresseurs algorithmiques** sont quant à eux basés sur l'exploitation de régularités algorithmiques et pas uniquement statistiques. De plus les références doivent être systématiquement faites au texte et pas à une structure de donnée externe telle qu'un dictionnaire.

Comme le montre notre exemple introductif relatif au texte ATATATATATATATATATAT, un compresseur, qu'il soit statistique ou à dictionnaire, ne permet pas de modéliser de manière satisfaisante l'événement "duplication d'une partie" (ici duplication est entendu comme répétition de la partie juste à côté). Dans le cadre des compresseurs statistiques, il est improbable que la partie dupliquée influe sur la distribution de probabilités des symboles, et donc sur la compression. Une telle duplication n'est donc pas détectée. Dans le cadre des compresseurs à dictionnaire, la méthode ne tire pas profit de la duplication et code simplement l'événement comme une répétition. C'est-à-dire que si les deux parties identiques ne se trouvent pas l'une à côté de l'autre mais sont plus éloignées dans le texte, le codage de la séquence comprimée sera le même et par conséquent la structure du texte révélée sera plus pauvre que si l'événement duplication était pris en compte de manière plus directe.

La limite entre compresseurs à dictionnaires et compresseurs algorithmiques est assez floue. Les compresseurs algorithmiques ont un niveau de raffinement supplémentaire qui leur permet de mieux comprendre les régularités d'un texte. Nous ne pouvons cependant pas proposer un critère permettant une classification d'un compresseur algébrique comme un compresseur à dictionnaire

ou un compresseur algorithmique, la limite entre les deux étant relativement subjective. Par exemple, avec la méthode LZ77, les références sont faites dans le texte, même si le texte est en fait une manière de stocker le dictionnaire, on peut considérer qu'il s'agit alors d'une méthode de compression algorithmique. Nous avons cependant vu que le simple codage des répétitions ne donnait qu'une information partielle et ne révélait que très peu les régularités d'un texte. Pour obtenir des méthodes d'analyse plus fine, il est nécessaire de se donner un modèle où les régularités détectées ne se limitent pas aux répétitions. A partir de ce moment il n'est plus possible de référencer simplement des mots dans un dictionnaire, il faut également identifier le type d'opération qui est effectuée.

Des travaux se basant sur l'utilisation de compresseurs algorithmiques pour l'étude des séquences ont déjà été effectués, nous en proposons un bref aperçu dans la section 3.3.3.

Rapports avec les théories de l'information

Attardons nous maintenant à l'examen du rapport entre l'idée de compression et les théories de l'information que nous avons présentées dans la précédente section.

Compresseur statistique et théorie de l'information de Shannon. Si l'on se réfère au paragraphe sur la théorie de l'information de Shannon (section 3.2.2), il apparaît clairement que les compresseurs statistiques utilisent cette idée. En effet, cette théorie définit le contenu moyen en information d'un texte source basé sur la distribution de probabilité de l'alphabet sur lequel est écrit ce texte. Ce contenu moyen en information est plus précisément défini comme la somme des contenus en information de chaque symbole pondéré par sa probabilité d'apparition. C'est la notion d'**entropie**. L'entropie E_i du symbole i de probabilité p_i est donnée par :

$$E_i = -\log(p_i)$$

Si le log est en base 2, l'entropie E_i mesure le nombre de bits moyen pour coder le symbole i. L'entropie du texte source est alors donnée par :

$$E = -\sum_i p_i \log(p_i)$$

Une méthode de compression statistique tente de fournir un schéma de codage qui s'approche le plus possible de l'entropie du symbole (pour les régularités statistiques basées sur la fréquence des lettres).

Compresseurs algorithmiques et théorie de la complexité de Kolmogorov

La notion de compresseur algorithmique est quant à elle reliée à la théorie algorithmique de l'information (section 3.2.2).

Les codes (ou les textes comprimés) engendrés par des compresseurs algorithmiques peuvent être vus comme des programmes qui produisent la séquence initiale une fois qu'ils sont exécutés sur une machine. Si l'on reprend l'exemple du texte `ATATATATATATATATATAT`, un compresseur dont le modèle est la détection de répétitions successives pourra produire comme version comprimée "10×`AT`". L'exécution du programme "10×`AT`" sur une machine capable de comprendre l'opération × pourra reconstruire le texte initial. Un tel programme correspond à une description du texte

tel qu'il est défini par la théorie algorithmique de l'information. Les compresseurs algorithmiques permettent ainsi une approximation de la complexité de Kolmogorov (en fait une majoration).

La figure 3.5 représente le parallèle que nous pouvons établir entre :

1. le compresseur et l'algorithme capable de calculer la complexité de Kolmogorov,
2. la séquence comprimée et le plus court programme,
3. la machine exécutant les opérations du modèle du compresseur et une machine de Turing universelle.

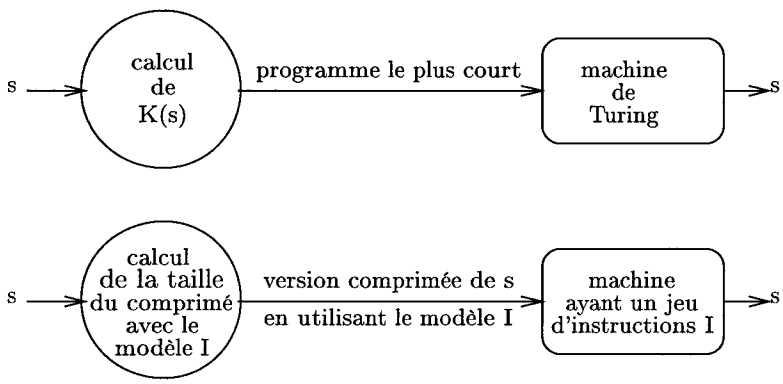


FIG. 3.5 – *Relation entre la signification du programme minimal pour s et la version comprimée de s d'une part, et $K(s)$ et le gain de compression d'autre part.*

3.3.2 Comprendre en comprimant

Nous avons déjà exprimé l'idée qu'une méthode de compression applique un modèle et que la mesure de la version comprimée du texte est une mesure de la pertinence du modèle par rapport au texte testé. Si un texte est bien comprimé, c'est-à-dire si le taux de compression est proche de 0, alors cela signifie que le modèle rend bien compte de la structure du texte. Cette compression a été possible grâce à la manière dont on a produit le texte comprimé. Dès lors, il apparaît clairement que la structure du texte correspond exactement au mécanisme utilisé pour la compression et donc que celui-ci représente un modèle pour la structure du texte.

Dans le cadre de l'analyse de séquences biologiques, des travaux ont déjà été effectués pour tenter de prouver cette idée que la compression est un bon moyen de comprendre un texte [RDDD96, GT93].

Textes compressibles

Nous voulons ici donner une idée de ce que sont des textes compressibles, et indiquer pourquoi et comment un compresseur peut permettre la compréhension d'un texte. Nous nous intéressons particulièrement aux séquences génétiques et présentons une succession de textes écrits sur l'alphabet des séquences d'ADN en donnant une version comprimée de ceux-ci et en expliquant le modèle sous-jacent. Débutons par une séquence triviale :

$$s_1 = \text{TTTTTTTTTTTTTTTTTTTTTTTTTTT}$$

Il s'agit ici d'une suite de lettres identiques, un code pour s_1 sera

"R(30,T)"

ce qui signifiera "répéter 30 fois T". Notre modèle est alors que les séquences sont des répétitions d'une même lettre. Pour écrire le texte comprimé, il faut alors donner le nombre de fois ou le symbole est répété et le symbole lui-même. '30' nécessite donc 13 bits pour être encodé (en utilisant le codage $x \mapsto 1^{|x|}0x$ discuté dans la partie sur l'auto-délimitation) et le symbole peut être codé sur 2 bits (l'alphabet est de taille 4), ce qui nous amène à un total de 15 bits. Donner la séquence explicitement nécessite 60 bits. Le taux de compression est alors de $1 - \frac{60-15}{60} = 0.25$ soit une réduction de 75 % de la taille initiale du texte. s_1 est très compressible, le modèle semble bien adapté. Intéressons nous maintenant à la séquence suivante :

$$s_2 = \texttt{TGATGATGATGATGATGATGATGATGA}$$

On distingue immédiatement qu'il s'agit d'une suite de triplets identiques, nous adoptons donc comme modèle qu'une séquence est la répétitions de quelques bases. Sous ce modèle, le codage de s_2 sera :

"R(10,TGA)"

Cette version de s_2 nécessite 21 bits, ce qui représente encore une fois un taux de compression très faible. Si nous avions codé la séquence s_2 avec le modèle utilisé pour s_1, le code aurait été :

"R(1,T), R(1,G), R(1,A), R(1,T), ... "

Nous voyons immédiatement que dans ce cas, il est plus intéressant d'écrire directement s_2 plutôt que d'écrire sa version encodée sous le modèle de s_1. Cela signifie que le modèle de s_1 n'apporte pas une bonne compréhension de la structure de s_2. Par contre s_1 peut être efficacement réécrite sous le modèle de s_2, cela nécessitera alors 18 bits. Le modèle de s_2 est plus général et permet la détection d'un plus grand nombre de régularités que celui de s_1 : il permet de décrire toutes les séquences qui sont des répétitions exactes d'un motif. L'application de ce modèle à une séquence permet grâce au taux de compression que l'on calcule de savoir si la séquence a plutôt une structure de répétitions exactes qu'une autre.

$$s_3 = \texttt{TGATATATGGATGGATGATATATGGATGGA}$$

Encore une fois, le modèle de s_1 ne fournira pas d'explication pertinente pour s_3, si nous appliquons le modèle de s_2, le codage de s_3 sera :

"R(2,TGATATATGGATGGA)"

ce qui nécessite 44 bits. Cette fois la compression est plus faible : 17 % seulement. Si nous regardons le texte de plus près, nous nous apercevons qu'en fait il peut être vu comme la répétition du motif TGGA avec quelques modifications, le codage devient alors :

"R(6,TGGA), I(2,ATA), S(3,G), I(18,ATA), S(19,G)"

où I(n,s) indique d'insérer le segment s en position n et S(n,x) indique de substituer la base se trouvant en position n par le symbole x. Ce codage se révèle en fait plus coûteux que le précédent (78 bits). Il est également plus sophistiqué puisqu'il permet de coder des erreurs par rapport à la description globale de la séquence. Le surcoût provient du fait qu'il y a de nombreuses mutations à coder par rapport à la longueur de la séquence. Une autre manière de coder cette séquence est :

"R(2, (R(3,TGGA), S(3,G), I(2,ATA))"

Ce codage nécessite quasiment le même nombre de bits que le codage basé sur le modèle des répétitions de s_2 (48 bits). Il est encore plus sophistiqué que le précédent mais explique aussi le mieux la structure de la séquence s_3.

Il apparaît que moins une séquence va être régulière, c'est-à-dire plus elle va contenir de mutations, plus il va être difficile de la comprimer fortement (avec un taux de compression faible). Si nous disposons d'une séquence tirée aléatoirement, il ne va pas être possible d'identifier une quelconque régularité. La meilleure manière de décrire la séquence est alors de la donner explicitement. Si l'on applique n'importe quel algorithme de compression, modélisant tous les types de régularités possibles, et qu'il n'est pas possible de comprimer la séquence alors on peut dire qu'elle est aléatoire.

Un compresseur peut donc aider, non seulement à la compréhension d'un texte mais également à faire la distinction entre des parties aléatoires et d'autres pour lesquelles il est possible d'extraire une information sur la structure [MJ93]. La difficulté à distinguer des séquences comportant des régularités et d'autres incompressibles s'accroît avec la complexité de la régularité étudiée. Le dernier modèle proposé pour s_3 qui est la répétition d'une autre régularité montre bien que la régularité d'une séquence n'est pas toujours facile à capter.

Un gain en compréhension peut être obtenu en cherchant à combiner différentes méthodes de compression. Il est possible d'observer pour un algorithme de compression la qualité de compression le long de la séquence. Cette observation peut être faite à partir d'une courbe reflétant le taux de compression par rapport à la position dans la séquence. On peut alors observer des pics aux endroits où la méthode compresse le mieux. Cela révèle par exemple les parties de séquences génétiques où il y a des répétitions en tandem répétées [Del97]. Il est alors possible de tenter de combiner plusieurs courbes en prenant pour chaque partie de la séquence la courbe offrant la meilleure compression. On obtient ainsi une combinaison d'algorithmes de compression et donc une combinaison de modèles décrivant le mieux possible la séquence complète. Si l'on dispose d'un ensemble de compresseurs suffisamment grand il est alors possible de bien caractériser une séquence.

Analyse statistique

Faisons une remarque sur la compression statistique des séquences dont nous venons de discuter. Nous avons vu que s_2 était plus compressible que s_3, et cela à cause d'une régularité moins remarquable dans s_3 que dans s_2. Si nous appliquons une méthode de compression statistique à s_2 et à s_3, nous obtiendrons exactement le même taux de compression. En effet, s_2 et s_3 sont composés des mêmes lettres et celles-ci sont présentent dans les mêmes proportions : 10 'A', 10 'T' et 10 'G'. Par conséquent un compresseur statistique ne sera pas capable d'identifier le type de régularité que le compresseur algorithmique que nous avons proposé a identifié. Cet exemple est très simple mais évoque toute la différence qu'il y a à observer des textes par une analyse statistique ou par une analyse algébrique (même si l'on peut envisager des régularités statistiques plus complexes).

Compression relative

Dans le cadre de la compression relative les mêmes arguments que précédemment peuvent être utilisés. Lorsqu'on comprime un texte par rapport à un autre on cherche des régularités

entre les deux textes. Par exemple, on peut dire que le second texte est le résultat du premier texte en effectuant une duplication :

$$s_1 = \texttt{ATTCATTAGGATGGAT}$$

et

$$t_1 = \texttt{ATTCATTACATTAGGATGGAT}$$

alors un schéma de codage pour t_1 par rapport à s_1 sera :

"DUP(3,5)"

où DUP(n,m) est la duplication du segment débutant à la position n et de longueur m. Dans ce cas, le codage nécessite 12 bits. Si maintenant j'ai deux séquences qui se ressemblent moins, alors le taux de compression va être plus élevé :

$$s_2 = \texttt{ATTCATTAGGATGGAT}$$

et

$$t_2 = \texttt{GTGGATGGATCCATTACCG}$$

Dans ce cas, il est difficile de trouver une régularité entre les deux séquences, alors que pourtant deux grands segments de t_2 peuvent être trouvés dans s_2 (`GGATGGAT` et `CATTA`).

Dans le cadre de la compression relative, le taux de compression permet, comme précédemment, de mesurer la pertinence d'un modèle. Ici le modèle détecte la structure relative du texte cible par rapport à celle du texte source. Par conséquent, le taux de compression permet une mesure de la ressemblance entre deux textes. Si les deux textes sont semblables alors il y aura des régularités entre les deux textes et donc il existera un compresseur capable de réécrire la séquence cible de manière plus courte et le taux de compression sera bon. Dans le cas contraire, les meilleurs compresseurs pour le texte cible le décriront quasiment en l'écrivant explicitement.

3.3.3 Analyse de séquences par compression

Nous décrivons brièvement quelques résultats obtenus dans le cadre de l'utilisation de la compression ou de méthodes de type compression pour l'étude de données génomiques.

Les premiers à avoir tenté de compresser des séquences génétiques se sont rendus compte de l'incapacité des programmes de compression de texte classiques à réduire la taille des séquences génétiques : la plupart du temps, le taux de compression est de 0.99, soit un gain de seulement 1% [Riv96] ! C'est d'autant plus étonnant que l'utilisation de ces compresseurs sur des textes "classiques", c'est-à-dire des textes en langue naturelle, des programmes, etc, fonctionnent bien et permettent le plus souvent d'atteindre un taux de compression de 0.5. Les raisons à cela sont multiples et décrites dans [Riv96]. On retiendra en particulier que le petit nombre de lettres qui compose l'alphabet d'une séquence d'ADN et la faible longueur des motifs répétés en sont deux causes bien identifiées.

Il a donc été nécessaire de construire des algorithmes spécifiques, capables d'exploiter les régularités des séquences génétiques. Le fait d'exploiter spécifiquement ces régularités implique que les résultats obtenus, en tant que taux de compression, peuvent être utilisés pour analyser une séquence par rapport au modèle mis en œuvre dans l'algorithme de compression.

Significativité algorithmique [MJ93].

Milosavljević et Jurka introduisent le concept de significativité algorithmique pour mesurer la simplicité d'une séquence. Une séquence est dite simple si elle peut être encodée en remplaçant les occurrences de segments par un pointeur vers la première occurrence de ce facteur. Un algorithme de programmation dynamique permet de rechercher la version compressée de longueur minimale obtenue avec ce schéma de codage. Le test de significativité permet ensuite de décider suivant la longueur de la séquence comprimée si elle est simple ou non.

Le concept de significativité algorithmique est le suivant. Soit s une séquence d'ADN et P_0 une distribution de probabilité sur les séquences supposant qu'elles sont générées aléatoirement. L'hypothèse nulle H_0 est que la séquence s a été générée aléatoirement selon la loi de probabilité P_0. Notons $l(A(s))$ la longueur binaire d'une version comprimée de s à l'aide de l'algorithme de compression A. Milosavljević et Jurka énoncent un théorème permettant de rejeter ou d'accepter l'hypothèse nulle H_0 avec un seuil de significativité 2^{-d} : *Pour toute distribution de probabilité* P_0 *et tout algorithme de compression* A, *on a*

$$\Pr(-\log(p_0(s)) - l(A(s)) \geq d) \leq 2^{-d}$$

où $p_0(s)$ *est la probabilité de la séquence* s *selon la loi* P_0 *et* $\Pr(E)$ *désigne la probabilité que l'événement* E *se produise.* Si s suit la loi de probabilité P_0, alors il est peu probable que la version comprimée de s soit d bits plus courte que $-\log(p_0(s))$. Si c'est le cas, alors l'hypothèse nulle est rejetée avec un seuil de significativité 2^{-d} et la séquence est dite simple.

Ce test de significativité algorithmique a été étendu dans [Mil93] pour être appliqué à la comparaison de séquences et à la mesure de la similarité de deux séquences. Encore une fois, les occurrences dans la séquence cible de segments communs à deux séquences source et cible sont remplacées par un pointeur sur l'occurrence dans la séquence source. L'hypothèse nulle est cette fois que la séquence cible est indépendante de la séquence source. Le test de significativité rejette ou admet cette hypothèse suivant la longueur du code décrivant la séquence cible à partir de la source.

BIOCOMPRESS [Tah97].

Grumbach et Tahi [GT93] ont mis au point l'algorithme BIOCOMPRESS. Cet algorithme comprime une séquence génétique en utilisant les répétitions exactes et les palindromes génétiques (c'est un algrithme de type LZ78). Le principe est un parcours de 5' vers 3' de la séquence. Pour chaque position envisagée, on recherche dans la partie de la séquence déjà parcourue le plus grand facteur ou le plus grand palindrome génétique concordant avec le segment débutant à la position courante. Si il y a un gain de compression, alors on code la répétition ou le palindrome par référence, comme ci-dessus.

Une amélioration a été proposée [GT94, GT95] en changeant le codage des bases en introduisant un codage arithmétique. Les gains de compression sont meilleurs et permettent l'identification de longs palindromes ou de longues séquences répétées.

Dans [Tah97] une version de BIOCOMPRESS dédiée à la comparaison de séquences est proposée. L'idée de la compression relative est utilisée comme dans le cas de la comparaison dans [Mil93].

CFACT et CPAL [Riv96].

L'inconvénient des méthodes type LZ78 (y compris BIOCOMPRESS) est qu'elles inspectent les séquences de gauche à droite. Dans la recherche de facteurs, cela a l'inconvénient de ne pas repérer des longs facteurs si ils apparaissent après la répétition d'un autre facteur.

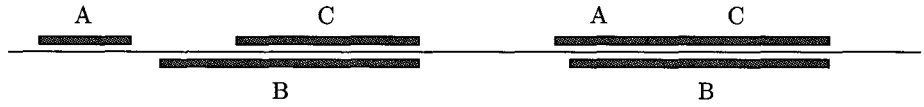


FIG. 3.6 – *Les algorithmes de compression basés sur LZ78 ne détectent pas le long facteur répété B car A et C ont été envisagés avant que B ne le soit.*

La méthode CFACT [Riv94] envisage les facteurs à encoder de manière différente. L'ensemble des facteurs repérés est trié par ordre de longueur décroissante et envisagé dans cet ordre. Pour chaque facteur, on regarde s'il est exploitable, c'est-à-dire s'il engendre une compression de la séquence. Si c'est la cas il est utilisé et chaque occurrence du facteur est remplacée par son codage. Il existe également une borne sur la longueur minimale des facteurs définie dans le cadre de la complexité de Kolmogorov. Seuls les facteurs de taille supérieure à $\log n$ sont conservés.

De plus, un théorème garantissant la compression de CFACT a été démontré.

Théorème 3.7 *Si la longueur du plus grand facteur répété dans la séquence $S \in \mathcal{N}^*$ est de longueur l et apparaît au moins deux fois aux positions i et j telles que $i + l < j$, alors la condition suivante est suffisante pour produire un gain de compression :*

$$2 \times l - |Fibo(l)| > 3 \times |Fibo(n)| + 1$$

où Fibo(x) représente le codage d'un entier x par le codage de Fibonacci [AF87].

L'algorithme CPAL est un équivalent de CFACT qui exploite les palindromes génétiques.

Une autre application est également décrite dans [Riv96] pour la détection des répétitions en tandem approximatives. Ce type de répétitions est formé de la répétition d'un même motif avec la contrainte qu'un motif peut également subir des mutations ponctuelles. Typiquement les motifs en question sont des di- ou trinucléotides.

CBI (Compression Based Induction) [LHYN95].

Cette fois le but est un peu différent des applications décrites auparavant. Etant donnés une séquence test et un ensemble de classes de séquences, on veut savoir à laquelle de ces classes la séquence test appartient.

L'idée décrite dans [LHYN95] est la comparaison de la séquence test à un échantillon de chacune des classes par une méthode de compression. La similarité est évaluée, et la séquence test est considérée comme appartenant à la classe associée à l'échantillon pour lequel la similarité est la plus grande.

L'évaluation est faite selon le processus suivant. Étant donné un algorithme de compression, C, on calcule le comprimé par C de la concaténation de l'ensemble des séquences d'un échantillon

et de la séquence test. Si l'échantillon est composé des séquences s_1,s_2,... ,s_n et la séquence test est t, on s'intéresse donc à la longueur

$$l = |C(s_1 s_2 \dots s_n t)|$$

On calcule ensuite

$$l' = |C(s_1 s_2 \dots s_n)|$$

La similarité entre la séquence test et l'échantillon est représentée par la différence

$$l - l'$$

Plus cette différence est petite, plus la similarité est grande et donc plus la séquence test est proche de la classe de l'échantillon considéré.

La classification obtenue dépend bien évidemment du compresseur utilisé. Les auteurs ont tenté des classifications en utilisant des types de compresseurs différents. Deux applications, la recherche de séquences promotrices et la reconnaissance des sites d'épissage, y sont abordées.

Alignement de séquences basé sur l'AIT [APD99].

Allison et *al.* [APD99] proposent une méthode d'alignement de séquences utilisant le point de vue de la théorie de l'information. La mesure du contenu en information d'un alignement est basée sur les différences entre les deux séquences et leur composition en bases.

Un des problèmes majeurs de l'alignement est qu'il peut être biaisé par le fait que les séquences possèdent elles-mêmes des biais statistiques. Par exemple des séquences peuvent avoir un bon score d'alignement uniquement parce qu'elles contiennent un fort taux de G+C. Il est donc difficile d'obtenir un alignement de bonne qualité, c'est-à-dire auquel on puisse accorder un grand degré de confiance, si les séquences comparées ont un faible contenu en information.

Les auteurs considèrent le script d'édition relatif à un alignement, c'est-à-dire la suite des opérations permettant la transformation d'une séquence en une autre. A chaque opération est affecté un poids qui prend en compte à la fois sa probabilité, mais également la probabilité de trouver le symbole mis en jeu.

- délétion et insertion jouant des rôles symétriques,

$$P(\langle S_i,- \rangle) = P(\text{delete}) \times P(S_i|S_{1..i-1})$$

$$P(\langle -,T_j \rangle) = P(\text{insert}) \times P(T_j|T_{1..j-1})$$

- copie correspondant à la conservation d'un symbole,

$$P(\langle S_i,T_j \rangle \,\&\, S_i = T_j = x) = P(\text{copy}) \times \frac{P(S_i = x|S_{1..i-1}) + P(T_j = x|T_{1..j-1})}{2}$$

- change correspondant à l'opération de substitution,

$$P(\langle S_i,T_j \rangle \,\&\, S_i = x \,\&\, S_j = y \,\&\, x \neq y)$$
$$= P(\text{change}) \times P(S_i = x|S_{1..i-1}) \times P(T_j = y|T_{1..j-1}) \times \frac{\frac{1}{1-P(S_i=y|S_{1..i-1})} + \frac{1}{1-P(T_j=x|T_{1..j-1})}}{2}$$

Cela permet d'accorder moins d'importance à des zones ayant un faible contenu en information. Avec le schéma suivant :

```
....AAAA....    ....ACGT....
    ||||            ||||
....AAAA....    ....ACGT....
     a)              b)
```

on préférera alors b) plutôt que a) si dans un alignement il n'est pas possible de conserver les deux.

Chapitre 4

Notions de script, mesures associées

Identifier et mesurer la ressemblance entre deux objets peut se faire de très nombreuses manières. L'une des plus naturelles est sans doute celle qui consiste à superposer les deux objets et à repérer leurs différences : c'est la méthodologie utilisée avec un alignement. Un reproche majeur que l'on peut faire à cette approche est qu'elle observe les différences entre deux objets mais ne tente pas d'identifier le processus ayant engendré ces différences. Une approche de type géométrique, c'est-à-dire recherchant une suite de transformations effectuant le passage de l'un à l'autre est sans doute plus indiquée lorsque l'on sait que les objets en question, ici des séquences biologiques, ont évolué par des événements les transformant. Chercher une suite de transformations permettant le passage d'un objet à un autre, ce que nous appelons un script, fournit une explication de la différence qui existe entre les deux objets, sous l'hypothèse que le modèle qu'on applique, c'est-à-dire les transformations elle-mêmes, reflètent effectivement des modifications possibles des objets. Nous définissons dans ce chapitre la notion générale de **script de transformation** et la mesure que l'on peut lui associer.

Un script est un procédé algorithmique permettant de déduire une **séquence cible** T à partir d'une **séquence source** S. Sommairement, un script peut être vu comme une série d'opérations appliquées successivement à S pour obtenir T. Chaque opération peut être interprétée comme une fonction capable de s'exécuter sur une machine particulière. Un script est ainsi un programme interprétable sur une machine disposant d'un jeu d'instructions correspondant à l'ensemble des opérations utilisées dans celui-ci. L'exécution d'un script sur une telle machine produira T lorsque S sera fournie en entrée.

A partir du script, il est possible de déduire une mesure de distance. Celle-ci est définie dans le cadre de la compression relative et, de manière plus formelle, dans le cadre de la théorie algorithmique de l'information. Nous verrons qu'un script est une version de T comprimée par rapport à S. La mesure de la longueur de cette version comprimée de T permet d'évaluer la quantité d'information partagée par S et T et donc le degré de similitude qu'il y a entre S et T. Il faut pour cela définir un schéma de codage permettant la mesure du coût (ou du poids) associé à chaque opération.

4.1 Introduction

Nous avons décrit dans la première partie les notions principales de ressemblance et de comparaison de séquences génétiques (chapitre 2). Toutes ces méthodes traitent principalement du problème de l'évaluation de la similarité, qu'elle soit qualitative, par la donnée d'une matrice de points par exemple, ou bien quantitative, par la mesure d'un score ou d'une distance. Nous avons vu que la majorité des comparaisons sont principalement basées sur la méthode d'alignement. Les méthodes d'alignement global restreignent leur application à la comparaison de séquences qui *a priori* sont très semblables. Dans des cas où les séquences sont trop éloignées, l'alignement n'a plus de signification car il est une succession de mutations ponctuelles pour aboutir finalement à un alignement similaire à celui de deux séquences aléatoires. Les méthodes où l'on effectue une recherche de similarités locales ne semblent pas satisfaisantes car elles limitent la recherche à une seule similarité, sans chercher à en mettre plusieurs en commun.

Des méthodes essayant de se détacher du principe de la comparaison base à base commencent à voir le jour. La méthode DIALIGN d'alignement basée sur l'identification de blocs similaires de Morgenstern et *al.* (section 2.1.6) entrevoit l'alignement comme la recherche de la mise en concordance de parties similaires. Même si les blocs considérés doivent respecter l'ordre naturel des séquences, il s'agit bien d'un élargissement de la définition d'alignement.

D'un autre côté, l'apparition de génomes complets demande de nouveaux outils pour la comparaison de séquences pour lesquels il est nécessaire de pouvoir considérer des blocs qui ne respectent pas l'ordre naturel des séquences. Nous pensons aux méthodes par réarrangement de segments que nous avons présentées section 2.1.7. L'inconvénient de ces méthodes est que le plus souvent un seul type d'opération à la fois est considéré. De plus, le poids associé à chaque opération est unitaire. Par conséquent, il n'y a pas de prise en compte de l'environnement (déplacement lointain ou proche, nombre de blocs mis en jeu dans une opération, etc.).

Dans la seconde partie, nous avons vu des outils mathématiques qui permettent une mesure de la ressemblance entre objets (chapitre 3). Si les méthodes décrites à la section 3.1 peuvent servir à une analyse de séquences biologiques, le concept de mesure de la quantité d'information par la complexité de Kolmogorov constitue quant à lui une voie nouvelle. Même si cette mesure absolue n'est pas calculable, nous avons vu qu'au travers des méthodes de compression nous pouvions en obtenir une approximation.

Nous présentons dans ce chapitre une notion de comparaison et une mesure de la similarité entre deux séquences génomiques dont les fondements reposent sur les idées issues de la théorie algorithmique de l'information. Nous pensons que la mesure de la quantité d'information partagée entre deux séquences est une bonne mesure de la ressemblance entre ces séquences. Si l'on considère que deux séquences partagent une grande quantité d'information, il est naturel de penser que ces séquences se ressemblent.

Mesurer la quantité d'information partagée nécessite de préciser ce que nous appelons information partagée. Deux séquences vont être considérées proches si il existe des régularités algorithmiques entre ces séquences, c'est-à-dire s'il existe un procédé algorithmique permettant de déduire une séquence d'une autre. Nous avons appelé ce procédé algorithmique **script**.

Dans un premier temps nous envisageons la définition formelle de script (section 4.2). Un script est défini comme une composition d'opérateurs capables de modifier une séquence : un

script effectue une **transformation** de S en T. Nous évoquons également le cas où la séquence T est calculée comme une reconstruction à partir de la séquence S. Nous verrons qu'une reconstruction est un cas particulier de transformation. Nous discutons des liens qui existent entre la notion de script et la notion de programme dans la théorie algorithmique de l'information, ainsi que la notion de compresseur dans le cadre de la compression de données. A partir de ce constat, nous déduisons une mesure de ressemblance entre deux séquences : la **distance de transformation**.

Le formalisme de la distance de transformation permet le calcul de distances connues entre séquences (section 4.3). Nous explorons les exemples de l'alignement et de la distance par inversions.

Nous nous attardons ensuite à la description d'un script basé sur deux opérateurs, l'un permettant d'extraire un segment d'une séquence et l'autre permettant de créer *ex-nihilo* un segment (section 4.4). Ce script a été mis en œuvre et constitue une implémentation de la distance de transformation. Les chapitres suivants traitent du problème algorithmique du calcul effectif de cette implémentation et de ses applications.

4.2 Notion de script

Appliquer une suite de transformations pour passer d'une séquence source à une séquence cible correspond à l'application d'une série de modifications apportées à des séquences intermédiaires. Ces modifications successives agissent sur le résultat de la modification précédente. Le schéma suivant décrit une série de n modifications, C_i est appelé la i-ème **séquence courante** et chaque m_i correspond à une opération de modification:

$$C_0 = S \xrightarrow{m_1} C_1 \xrightarrow{m_2} C_2 \xrightarrow{m_3} \ldots \xrightarrow{m_n} C_n = T$$

Ceci est le schéma global d'un script. Afin de rendre les choses plus formelles, nous allons préciser ce que sont, ou ce que peuvent être, les opérations de modification. Nous introduisons donc la définition d'opérateur, qui sera la brique de base permettant de construire des scripts, puis la définition de script comme composition d'opérateurs élémentaires.

Notation 4.1 *On notera :*

- Σ *l'alphabet sur lequel sont écrites les séquences source et cible,*
- N *l'ensemble des entiers naturels,*
- Z *l'ensemble des entiers relatifs.*

4.2.1 Opérateurs

La notion d'**opérateur** correspond à la notion abstraite de modification évoquée juste avant. Un opérateur agit sur une séquence pour la modifier, en extraire une partie, en supprimer une autre, etc. Un opérateur transforme une séquence en fonction de paramètres pour produire une autre séquence.

Exemple 4.1 *L'opérateur* `repet(`s`,`n`)` *construit la séquence composée de n séquences s :*

`repet(ATC,3) = ATCATCATC`

L'opérateur **seq**(*s,p,l*) *extrait un segment de s à partir de la position p et d'une longueur l :*

$$\texttt{seq(GATTGGATGCCAGAT,4,5)} = \texttt{TGGAT}$$

La séquence que l'opérateur transforme est dénommée sa **variable**, les autres inconnues sont les **paramètres**. Un opérateur est donc une application dans Σ^*. L'ensemble de départ varie avec la nature de l'opérateur mais il doit systématiquement y apparaître la variable. Seul le nombre et la nature des paramètres est fonction de l'opérateur.

Nous supposerons qu'un opérateur a toujours connaissance de la longueur de la séquence courante.

Classes d'opérateurs et opérateurs instanciés. Nous distinguons les opérateurs en deux catégories. De manière générale, nous allons définir des **classes d'opérateurs**. Ces opérateurs correspondent à une spécification générique d'une méthode transformant une séquence. Ils désignent une classe d'opérateurs.

Exemple 4.2 *On pourra disposer d'une classe d'opérateurs permettant d'effectuer une duplication d'un segment débutant en position p et de longueur l en position p + l. Un tel opérateur sera défini comme suit:*

$$\begin{array}{rccl} dup & : & \Sigma^* \times N \times N & \longrightarrow & \Sigma^* \\ & & (u,p,l) & \longmapsto & u[0..p].u[p,l].u[p,l].u[p+l..|u|] \end{array}$$

Cet opérateur désigne l'ensemble des opérateurs dup.

Si maintenant on précise les paramètres d'une classe d'opérateurs, alors on instancie cet opérateur. Les paramètres sont fixés une fois pour toutes. On parle alors d'**opérateur instancié**.

Exemple 4.3 *Une instance de l'opérateur* **dup** *sera la duplication du segment compris entre les positions 10 et 10+5 :*

$$\begin{array}{rccl} dup_{10,5} & : & \Sigma^* & \longrightarrow & \Sigma^* \\ & & u & \longmapsto & u[0..10].u[10,5].u[10,5].u[15..|u|] \end{array}$$

Un opérateur instancié est une application de Σ^* dans Σ^*. Dans la suite, lorsque l'on parlera d'opérateur on fera toujours référence à des opérateurs instanciés. Dans le cas contraire on précisera qu'il s'agit de classes d'opérateurs.

Remarque 4.1 *Il se peut qu'une classe d'opérateurs se confonde avec ses instances. Cela se produit si la classe ne dispose pas d'un ensemble de paramètres.*

Notation 4.2 *Pour simplifier l'écriture de la suite, si o représente une classe d'opérateurs, on notera* $\overline{o}$ *une de ses instances.*

Composition. L'opération de composition ne concerne que les opérateurs instanciés. Elle est définie tout comme la composition de fonctions. Si $\overline{o}_1$ et $\overline{o}_2$ sont deux opérateurs instanciés, $\overline{o}_1 \circ \overline{o}_2(u) = \overline{o}_1(\overline{o}_2(u))$. La composition d'opérateurs permet ainsi de définir d'autres opérateurs.

Exemple 4.4 *Imaginons ne disposer que de la seule classe d'opérateur dup. Il est possible de créer des instances qui par composition vont permettre une double duplication:*

$$dup_{2,3} \circ dup_{2,3}(ATGGGAT) = dup_{2,3}(dup_{2,3}(ATGGGAT)) = dup_{2,3}(ATGGGGGGAT) = ATGGGGGGGGGAT$$

Système générateur. Nous introduisons la notion de système générateur pour désigner un ensemble de classes d'opérateurs. Cet ensemble contient les opérateurs dont on pourra composer les instances pour obtenir un script permettant de calculer le passage de S à T.

Exemple 4.5 *Si ins est l'opérateur défini de la manière suivante:*

$$\begin{array}{rccc} ins & : & \Sigma^* \times N \times \Sigma^* & \longrightarrow & \Sigma^* \\ & & (u,p,v) & \longmapsto & u[0..p].v.u[p+l..|u|] \end{array}$$

alors $\mathcal{G} = \{ins,dup\}$ sera un système générateur à partir duquel il sera possible de construire des instances de ins et dup et de les composer.

4.2.2 Définition du script

Une fois ces définitions données, il nous est possible de définir formellement un script, comme la composition d'opérateurs instanciés.

Définition 4.1 *Soit $\mathcal{G}_t$ un système générateur. Un* ***script de transformation*** *$(\overline{m}_1,\overline{m}_2,\ldots,\overline{m}_n)$, $m_i \in \mathcal{G}_t$ est une application qui à toute séquence $s \in \Sigma^*$ associe $\overline{m}_n \circ \overline{m}_{n-1} \circ \ldots \circ \overline{m}_1(s)$.*

Remarquons qu'un script peut être interprété comme un unique opérateur instancié composé d'autres opérateurs instanciés.

Remarque 4.2 *Notons qu'il n'existe pas forcément de script capable de transformer une séquence donnée S en une autre séquence donnée T. Cela dépend du système générateur. Prenons l'exemple d'un système générateur ne contenant que la classe d'opérateurs capable de dupliquer une partie d'une séquence. Il n'existe pas dans un tel système générateur d'opérateur capable de produire des parties non communes à deux séquences. Pour être certain de produire toute séquence T à partir de toute séquence S il est nécessaire que le système générateur dispose d'au moins un opérateur générant des séquences qui ne sont pas communes à S et T.*

4.2.3 Reconstruction

La notion de script que nous venons de définir correspond à une idée naturelle que l'on a si l'on se pose la question de décrire un objet à partir d'un autre. De manière intuitive on choisira de dire qu'il suffit d'effectuer tel et tel types de modifications pour aboutir à l'objet cible à partir de l'objet source. Cela correspond à une idée de transformation.

Mais une vue différente peut être proposée. La description d'une séquence cible à partir d'une séquence source peut se faire également grâce à un processus de reconstruction de la séquence cible. En débutant avec une séquence courante vide (ne contenant rien), on ajoute successivement des segments t_i pour aboutir finalement à la séquence cible :

$$\varepsilon \longrightarrow t_1 \longrightarrow t_1 t_2 \longrightarrow \ldots \longrightarrow t_1 t_2 \ldots t_n$$

Contrairement au script de transformation, c'est une opération de concaténation du résultat de la modification i à la $(i-1)$ème séquence courante qui permet d'obtenir T à partir de S. C'est-à-dire qu'à la séquence courante C_{i-1} on ajoute un segment t_i, celui-ci étant le résultat d'un opérateur. Cet opérateur n'a pas connaissance de la séquence C_{i-1}. Il y a donc indépendance des résultats des différents opérateurs, qui comme le montre la définition que nous donnons ci-dessous, ne dépendent que de la séquence source.

Remarquons que le segment t_i peut être lui-même le résultat d'une composition d'opérateurs instanciés. C'est-à-dire que chaque segment t_i de la concaténation peut lui-même être le résultat d'un opération complexe. L'exemple suivant illustre cela.

Exemple 4.6 *Soit un système générateur comprenant les opérateurs suivants :*

$$\begin{array}{rcccc} inv & : & \Sigma^* & \longrightarrow & \Sigma^* \\ & & u & \longmapsto & u^{-1} \end{array}$$

$$\begin{array}{rcccc} dup & : & \Sigma^* & \longrightarrow & \Sigma^* \\ & & u & \longmapsto & u.u \end{array}$$

$$\begin{array}{rcccc} cop & : & \Sigma^* \times N \times Z & \longrightarrow & \Sigma^* \\ & & (u,p,l) & \longmapsto & u[p..p+l-1] \end{array}$$

Un segment de la concaténation peut très bien être obtenu grâce à la composition d'opérateurs instanciés suivante :

$$\overline{dup} \circ \overline{inv} \circ \overline{cop}_{3,5}(S)$$

qui correspond à la duplication du segment de S, inversé, débutant en position 3 et de longueur 5. Ainsi, si $S = ATcgtaaGTA$, le résultat de l'application de la composition d'opérateurs précédente sera aatgcaatgc :

$$\begin{aligned} \overline{cop}_{3,5}(ATcgtaaGTA) &= cgtaa \\ \overline{inv} \circ \overline{cop}_{3,5}(ATcgtaaGTA) &= \overline{inv}(cgtaa) \\ &= aatgc \\ \overline{dup} \circ \overline{inv} \circ \overline{cop}_{3,5}(ATcgtaaGTA) &= \overline{dup}(aatgc) \\ &= aatgcaatgc \end{aligned}$$

Si $T = ATaatgcaatgcGTA$, un script de reconstruction de T à partir de S sera :

$$\overline{cop}_{1,2}(S).\left[\overline{dup} \circ \overline{inv} \circ \overline{cop}_{3,5}(S)\right].\overline{cop}_{8,3}(S)$$

Nous pouvons maintenant proposer une définition pour un script de reconstruction.

Définition 4.2 *Soit $\mathcal{G}_r$ un système générateur. Un* ***script de reconstruction*** *$(\overline{r}_1, \overline{r}_2, \ldots, \overline{r}_n)$, $\overline{r}_i = \overline{r}_{i,k_i} \circ \overline{r}_{i,k_i-1} \circ \ldots \circ \overline{r}_{i,1}$, $k_i \geq 1$ $\forall i \in 1..n$, $r_{i,j} \in \mathcal{G}_r$, est l'application qui à toute séquence s de Σ^* fait correspondre $\overline{r}_1(s).\overline{r}_2(s)\ldots\overline{r}_n(s)$.*

Par conséquent, les segments de T impliqués dans une reconstruction ne peuvent pas se chevaucher dans T. Rien n'empêche en effet de se servir plusieurs fois d'une même portion de S pour reconstruire T mais il est impossible que deux des t_i utilisés ci-dessus se chevauchent. Cette propriété sera fort utile lorsqu'il s'agira de chercher un script de reconstruction.

Propriété 4.1 *Comme T est le résultat d'une concaténation de segments, il est possible de segmenter la séquence cible et d'obtenir chaque partie indépendamment des autres. Si l'on dispose d'une reconstruction $\overline{r}_1,\overline{r}_2,\ldots,\overline{r}_n$ de T à partir de S alors il existe une suite $t_1,t_2,\ldots,t_n$ de segments de T, tels que*

$$T = t_1 \ldots t_n$$

où chaque t_i est égal à $\overline{r}_i(S)$.

Une reconstruction comme une transformation

Clairement une reconstruction n'est pas une transformation. En effet, d'un côté il s'agit d'une concaténation de résultats d'opérateurs alors que de l'autre il s'agit d'une composition d'opérateurs. Mais fondamentalement, l'idée n'est pas différente. Il est possible d'interpréter une reconstruction comme une composition d'opérateurs particuliers en redéfinissant les opérateurs du système générateur de manière un peu différente. Comme chaque opérateur dispose toujours de la même variable, S, il est possible de définir d'autres opérateurs ayant un paramètre instancié à S.

Nous détaillons les opérateurs permettant de faire cela maintenant. Posons

$$\rho_i(Y,X) = \overline{r}_i(X).Y$$

On s'intéresse à l'instance de ρ_i définie par

$$\overline{\rho}_i^S(Y) := \rho_i(Y,S) = \overline{r}_i(S).Y$$

On a alors:

$$\overline{\rho}_i^S \circ \overline{\rho}_j^S(Y) = \overline{\rho}_i^S(\overline{\rho}_j^S(Y)) = \overline{\rho}_i^S(\overline{r}_j(S).Y) = \overline{r}_i(S).\overline{r}_j(S).Y$$

On peut maintenant définir T comme le résultat du script $(\overline{\rho}^S{}_1,\overline{\rho}^S{}_2,\ldots,\overline{\rho}^S{}_n)$ appliqué à ε:

$$\begin{array}{rcccccccc} T & = & \overline{\rho}_1^S & \circ & \overline{\rho}_2^S & \ldots & \overline{\rho}_n^S & (\varepsilon) & \\ & = & \overline{r}_1(S) & . & \overline{r}_2(S) & \ldots & \overline{r}_n(S) & . & \varepsilon \\ & = & t_1 & . & t_2 & \ldots & t_n & & \end{array}$$

Nous aboutissons ainsi à une définition différente, mais équivalente, d'un script de reconstruction vu comme une transformation construite sur un système générateur particulier: un script de reconstruction $(\overline{r}_1,\overline{r}_2,\ldots,\overline{r}_n)$ est alors l'application qui à toute séquence S de Σ^* fait correspondre $\overline{\rho}^S{}_1 \circ \overline{\rho}^S{}_2 \circ \ldots \circ \overline{\rho}^S{}_n(\varepsilon)$ avec $\overline{\rho}_i^S(u) = \overline{r}_i(S).u$ pour tout mot u de Σ^*.

Cela montre que la définition d'un script de reconstruction est très proche d'un script de transformation. La différence se situe au niveau de l'interprétation du script calculant T à partir de S. L'opérateur ρ que nous définissons ici est particulier puisqu'en fait il ne transforme pas sa variable mais lui ajoute une séquence. L'idée sous-jacente que l'on associe au terme de transformation est la modification en tant que remaniement. C'est-à-dire que l'on attend d'un processus de transformation qu'il recompose l'objet sur lequel celle-ci est réalisée. Si l'ajout d'une partie à

un objet est bel et bien une transformation, il ne doit pas y avoir confusion quant à la nature de la démarche que l'on a lorsqu'on effectue des ajouts successifs. Un processus de reconstruction ne cherche pas à imiter un quelconque processus biologique, puisqu'il s'agit ici d'objets biologiques, capable de construire une nouvelle séquence à partir d'une autre.

4.2.4 Poids d'un script

Nous montrons dans cette section pourquoi il est possible de déduire une mesure de la ressemblance entre S et T à partir d'un script de transformation. Nous nous appuyons sur des considérations théoriques évoquées au chapitre précédent. Dans un premier temps nous exhibons la similitude qui existe entre un script et le résultat d'un algorithme de compression en tant que méthode descriptive. Nous montrons ensuite que la mesure de la longueur d'un script fournit une approximation de la complexité de Kolmogorov relative entre S et T.

Une première approche pourrait être de compter le nombre d'opérations de transformation contenues dans un script permettant de calculer T à partir de S. Une telle mesure n'est pas satisfaisante, pour plusieurs raisons. Tout d'abord, si nous supposons travailler avec un système générateur possédant un opérateur capable de créer des séquences, nommons le `créer`, le script le plus court permettant de passer de S à T sera "`créer(T)`". Il n'existera en effet pas de plus petit script, c'est-à-dire de script formé de moins d'une opération. Dans le cas contraire où une telle opération n'existe pas, la longueur du script mesure donc un nombre de pas de transformation pour obtenir T à partir de S.

Une telle mesure ne reflète que très faiblement le degré de similitude de deux séquences. En effet, cela ne prend pas en compte les probabilités que certaines opérations se produisent plutôt que d'autres. Il n'y a pas non plus de prise en compte de l'environnement dans lequel s'effectue une opération.

Un script comme la version comprimée d'une séquence

Nous avons défini un script comme la composition d'un certain nombre d'opérateurs. Cette manière de définir une séquence cible à partir d'une séquence source correspond à un mode de description de la seconde séquence à partir d'informations extraites de la première. En ce sens un script est comme une façon de réécrire la séquence cible. Un script peut être interprété comme une version comprimée de la séquence cible, et l'algorithme qui produit le script comme un compresseur. De plus un tel algorithme entre dans la classe des compresseurs algorithmiques. En effet, la définition d'un opérateur n'impose aucune contrainte, il est donc possible de définir des opérateurs complexes, capables de se composer pour exhiber des régularités non triviales.

Le système générateur sur lequel est défini un script correspond au jeu d'instructions, c'est-à-dire au modèle, que le compresseur peut utiliser pour construire des versions comprimées de T. Cet ensemble d'instructions est forcément réduit. Par conséquent, la recherche d'un script écrit dans un système générateur particulier permettra la mise en avant de certaines propriétés, relatives aux opérateurs du système générateur. Par exemple, si le système générateur contient un opérateur permettant les duplications de parties d'une séquence, alors un script écrit dans ce système exhibe cette propriété de segments dupliqués entre la séquence source et la séquence cible. Remarquons que même si la composition de deux ou plusieurs duplications permet des répétitions successives, le script n'évaluera que des duplications et pas d'autres opérations. Nous

voulons exprimer ici que le fait de combiner des opérations permet de détecter des successions d'un même événement mais pas la construction d'un nouvel événement en tant que tel. Détecter qu'un événement se produit plusieurs fois n'est pas équivalent à détecter un seul événement répétitif.

Le choix d'un système générateur modifie donc de manière notable le critère de ressemblance.

Si par contre on disposait d'un système générateur universel, c'est-à-dire contenant tous les opérateurs possibles (qui correspondent à toutes les fonctions calculables) alors on disposerait d'un outil capable d'exhiber n'importe quelle propriété entre deux séquences, et donc les régularités exhibées seraient celles correspondant à la meilleure description, universelle, de la séquence source à partir de la séquence cible. Il est alors possible d'établir un rapport étroit entre le système générateur et une machine de Turing universelle d'une part, et le script et le programme qui s'exécute sur cette machine de Turing d'autre part. Le script correspond à la description d'un texte tel qu'il est envisagé dans la théorie algorithmique de l'information (voir section 3.3).

La longueur d'un script comme approximation de la complexité relative de Kolmogorov

Admettons maintenant que nous écrivions notre script dans le langage binaire. Nous pouvons définir la **longueur du script** comme le nombre de bits nécessaires à son écriture (comme les textes et les programmes que nous avons considérés dans la section 3.2). Si nous disposions d'un système générateur universel, alors la longueur du script le plus court correspondrait exactement à $K(T|S)$. En effet, avec un système générateur universel tous les scripts possibles peuvent être produits, et en particulier celui dont la longueur est la plus courte.

Un système générateur possédant un pouvoir de calcul plus faible (ne pouvant pas calculer l'ensemble des fonctions partiellement récursives) ne pourra que très exceptionnellement produire le script de longueur minimale. C'est le cas dans la pratique où nous ne disposons que d'un système générateur possédant un nombre limité d'opérateurs, la recherche du script le plus court produit par ce système générateur ne fournit qu'une majoration de la complexité de Kolmogorov de T relativement à S.

Dans ce cadre, la longueur du script le plus court permet une mesure de la similarité qui existe entre la séquence source S et la séquence cible T relativement au système générateur sur lequel le script est écrit.

Le choix du système générateur influence la notion de similarité définie. La mesure décrite ici est largement dépendante du système sur lequel est basé le script. Un système générateur contenant un opérateur permettant les inversions produira potentiellement des scripts capables de les détecter alors qu'un système générateur ne le contenant pas (et tel qu'aucune composition ne le simule) sera incapable de les détecter. La mesure que nous proposons n'est donc pas universelle. Elle reflète l'application d'un modèle, qui est l'ensemble des opérations autorisées pour effectuer la transformation de S en T. L'approche par la théorie algorithmique de l'information et, pour le côté applicatif, par la compression de données permet de fournir une meilleure explication pour un modèle donné [RDDD96]. Si des régularités sont identifiées pour un système générateur, elles

correspondent à une optimisation pour ce système générateur, c'est-à-dire à une compression optimale. Le script le plus court correspond au script minimisant la quantité d'information qu'il faut ajouter à la séquence S pour obtenir la séquence T.

Remarque 4.3 *La notion de recherche d'une plus courte description correspond à la notion de parcimonie bien connue, notamment en phylogénie (voir section 2.2). L'hypothèse a priori que l'évolution est parcimonieuse implique que l'on préférera un arbre contenant un nombre minimal de pas évolutifs à un arbre en contenant plus. Dans notre cas, lorsque nous préférons un script plus court, nous nous référons au principe du rasoir d'Occam (voir section 3.2.3) : les explications les plus courtes sont les meilleures (en l'absence d'informations complémentaires).*

Schéma de codage

Il apparaît maintenant clairement que l'évaluation d'une ressemblance à partir d'un script est obtenue par la mesure de la longueur du script. Nous avons vu que nous pouvions mesurer la longueur d'un script par la longueur de son écriture dans le langage binaire. Pour obtenir cette écriture, il est nécessaire de se donner un **schéma de codage** tout comme nous devons le faire lorsque nous désirons écrire une version comprimée d'un texte. Il faut donc associer à chaque classe d'opérateurs d'un système générateur une fonction de codage donnant son écriture dans le langage binaire.

Remarque 4.4 *Remarquons qu'il est nécessaire que tous les opérateurs soient codés dans le même langage si l'on veut pouvoir comparer les longueurs des codages. Le choix du langage binaire ne va pas de soi, tout autre langage peut convenir. Cependant nous avons choisi celui-ci pour sa simplicité et la connaissance que nous en avons par son utilisation systématique dans les méthodes de compression. Son choix est naturel dans le contexte des scripts de transformation.*

Une fois un langage choisi, nous pouvons mesurer la longueur du codage de chaque opérateur comme le nombre de lettres utilisées pour décrire l'opérateur sur l'alphabet du langage choisi. Nous appellerons **poids** ou **coût** cette longueur. Par exemple, il faut définir la façon dont sera écrite une opération de duplication lors de la réécriture de la séquence cible à partir de la séquence source. En langage naturel, nous dirions "il faut dupliquer la partie allant de la position x à la position $x + p$", et cela pourrait être un codage. Plus précisément, nous devons indiquer : i/ que c'est une opération de duplication, ii/ la position x du segment à dupliquer et iii/ la longueur p du segment à dupliquer. La définition d'un schéma de codage nécessite donc de donner, pour chaque opérateur, une suite d'**attributs**, c'est-à-dire sa description, et le **codage** de ceux-ci dans le langage choisi.

Comme notre but est d'obtenir le script le plus court, le choix d'un opérateur pour représenter une partie de la séquence cible correspond au choix d'un opérateur qui décrit cette partie de manière la plus courte.

Le choix du schéma de codage, autant que le choix du système générateur est important dans l'identification de la ressemblance que nous voulons effectuer.

Admettons que nous disposions d'un schéma de codage optimal pour un système générateur donné. C'est-à-dire que chaque opérateur est codé de telle manière qu'il n'existe pas de représentation plus courte. Dans ce cas, aucun des opérateurs du système générateur n'est avantagé par

rapport à un autre. En effet, comme les opérateurs ont une représentation optimale, l'utilisation de tel ou tel opérateur dans la réécriture d'une partie de la séquence cible implique que pour coder cette partie, c'est précisément cet opérateur qui est le plus approprié puisqu'il va minimiser la longueur de la représentation. Par conséquent nous choisirons cet opérateur.

Il n'existe pas de définition standard et unique de l'optimalité d'un codage. Rivals et Delahaye ont travaillé à la définition d'une notion d'optimalité en moyenne [RD98]. Dans les scripts que nous allons écrire, nous n'aurons certainement pas la propriété qu'un schéma de codage est optimal. Par conséquent, lorsqu'un opérateur sera choisi plutôt qu'un autre pour représenter une partie de la séquence cible, ce sera parce que sa représentation, dans le schéma de codage que nous avons défini, sera plus courte. Cela a l'indéniable avantage de la modularité. Nous pourrons en effet faire que tel ou tel opérateur sera plus ou moins bien codé, et donc d'une représentation plus ou moins optimale, pour exhiber des propriétés que nous recherchons. Il sera ainsi possible de prendre en compte des connaissances particulières que nous avons sur les données que nous devons traiter. A l'opposé, le désavantage majeur est l'introduction d'une certaine subjectivité quant à la définition de la notion de ressemblance.

4.2.5 Problème général de la distance de transformation

Le problème que nous voulons résoudre peut être désormais clairement formulé.

Notons ω le poids d'un opérateur. La longueur du script, aussi appelé poids ou coût du script, est la somme des poids des opérateurs. Si $(\overline{m}_1,\overline{m}_2,\ldots,\overline{m}_n)$ est un script, le poids Ω de celui-ci est défini ainsi :

$$\Omega(\overline{m}_1,\overline{m}_2,\ldots,\overline{m}_n) = \sum_{i=1}^{n} \omega(\overline{m}_i)$$

Définition 4.3 *Etant donnés un système générateur $\mathcal{G}_t$, deux séquences S et T, un schéma de codage associé à $\mathcal{G}_t$, un script de transformation φ sur $\mathcal{G}_t$ de S en T est dit* ***minimal*** *si il n'existe pas de script ϕ sur $\mathcal{G}_t$ permettant la transformation de S en T tel que*

$$\Omega(\phi) < \Omega(\varphi)$$

Le problème que nous nous posons est donc la recherche d'un script de transformation de S en T qui soit minimal.

Problème 4.1 (Problème de la distance de transformation) *Etant donnés un système générateur $\mathcal{G}_t$, une séquence S et une séquence T, un schéma de codage associé au système générateur $\mathcal{G}_t$, trouver un script $(\overline{m}_1,\overline{m}_2,\ldots,\overline{m}_n)$ tel que $\overline{m}_1 \circ \overline{m}_2 \circ \ldots \circ \overline{m}_n(S) = T$ et $\Omega(\overline{m}_1,\overline{m}_2,\ldots,\overline{m}_n)$ soit minimal.*

Définition 4.4 *$td(S,T)$ désigne le coût de la transformation minimale de S en T*

Interprétation

Nous avons expliqué dans la section précédente et l'introduction de ce chapitre les motivations qui nous ont amené à définir la distance de transformation : *la définition d'une notion générale*

de mesure de la ressemblance basée sur la mesure directe de l'information partagée entre deux objets.

Le résultat d'un calcul de similitude par la distance de transformation ne reflète que l'optimisation d'une méthode de compression. Même si nous défendons l'idée que le résultat d'une compression permet de comprendre, que c'est une sorte d'explication de l'évolution entre la séquence source et la séquence cible, nous ne faisons en aucun cas référence à l'Evolution des espèces. Le résultat obtenu par la distance de transformation est une proposition, qui, relativement au modèle choisi, c'est-à-dire le système générateur, et à l'importance donnée aux différents opérations, c'est-à-dire le schéma de codage, est basée sur l'optimisation de la fonction définie ci-dessus.

Si le modèle que nous avons défini pour une méthode particulière implémente des opérations ayant une pertinence biologique, alors il est possible, pour le biologiste, de tirer des conclusions du script obtenu. Nous ne pouvons affirmer que la simulation de la transformation d'une séquence en une autre au travers d'un script est la reconstitution du schéma évolutif qui a permis, au cours du temps, d'aboutir à la séquence T à partir de la séquence S.

L'approche que nous proposons de manière générale est la définition de modèles biologiques que nous adaptons pour la compression de séquences. Les résultats obtenus à partir de tels modèles peuvent avoir des implications biologiques et aider à la compréhension et l'analyse des données.

4.3 Des systèmes générateurs connus

Il est possible de définir des systèmes générateurs qui permettent le calcul d'une mesure de dissimilarité connue entre séquences génétiques. Nous montrons cela sur deux exemples : d'une part pour le calcul du score d'un alignement (voir le paragraphe 2.1.4), et d'autre part pour le calcul du nombre minimum d'inversions entre deux génomes (voir le paragraphe 2.1.7). Les schémas de codage définis pour les opérateurs ne sont pas en relation avec le contenu en information de ceux-ci, au sens où nous l'avons défini précédemment. Nous désirons simplement illustrer le fait que le problème de la distance de transformation englobe un certain nombre de problèmes de calcul de similarité.

4.3.1 Le système générateur ALIGN

Lorsque l'on effectue un alignement, on calcule un score grâce à une équation de récurrence donnée page 26 [NW70]. Il est possible d'effectuer trois opérations de mutations : insertion d'une base, délétion d'une base et substitution d'une base par une autre. Une quatrième opération implicite est la conservation d'une base. Cette opération est effective dans le calcul du score d'un alignement puisqu'il est nécessaire de connaître le nombre de bases non mutées pour obtenir sa valeur. Aussi, si l'on interprète un alignement comme une transformation, il ne faut pas se contenter de citer la liste des opérations de mutations mais également la liste des opérations identité sans laquelle on ne dispose pas de toute l'information pour aboutir au calcul du score tel qu'il est défini par l'équation de récurrence. Il s'ensuit que pour permettre le calcul d'un alignement par un script de transformation, le système générateur doit comprendre quatre classes d'opérateurs :

$$\mathcal{G}_{\text{ALIGN}} = \{ins,\ del,\ sub,\ id\}$$

où *ins, del, sub* et *id* sont définis comme suit ($\mathcal{N}$ représente l'alphabet des nucléotides) :

$$\begin{array}{rcccc}
ins & : & \mathcal{N}^* \times N \times N & \longrightarrow & \mathcal{N}^* \\
 & & (s,p,b) & \longmapsto & s[1..p].b.s[p+1..|s|] \\
del & : & \mathcal{N}^* \times N & \longrightarrow & \mathcal{N}^* \\
 & & (s,p) & \longmapsto & s[1..p-1].s[p+1..|s|] \\
sub & : & \mathcal{N}^* \times N \times N & \longrightarrow & \mathcal{N}^* \\
 & & (s,p,b) & \longmapsto & s[1..p-1].b.s[p+1..|s|] \\
id & : & \mathcal{N}^* \times N & \longrightarrow & \mathcal{N}^* \\
 & & (s,p) & \longmapsto & s
\end{array}$$

Si les coûts associés aux substitutions, aux gaps et à l'identité sont respectivement -1, -2, 0, alors les fonctions de coût associées aux classes d'opérateurs sont :

$$\omega(ins(s,p,b)) = 2$$

$$\omega(del(s,p)) = 2$$

$$\omega(sub(s,p,b)) = 1$$

$$\omega(id(s)) = 0$$

Il est nécessaire de prendre l'opposé des coûts classiques utilisés pour l'alignement car le score associé à un alignement est un score de similarité alors que la distance de transformation mesure la dissimilarité. On aura ainsi :

$$\text{score}(S,T) = -\text{td}(S,T)$$

Notons que les schémas de codage adoptés ne reflètent pas la quantité d'information nécessaire à la description de chaque opération. Il est concevable de ne pas considérer les positions dans les codages puique l'alignement peut être vu comme une reconstruction, où l'on applique successivement les opérations de mutations ponctuelles. Par contre, considérer un poids identique à l'insertion et la délétion n'est pas justifié, d'un point de vue informationnel, puisqu'il est nécessaire de préciser la base à insérer pour l'opération d'insertion.

Exemple 4.7 *Soit* $S =$ `ACTGAT` *et* $T =$ `AGGATT`. *Un script reflétant l'alignement suivant :*

$$\begin{array}{rl}
u = & \texttt{ActGA-T} \\
 & \texttt{:\ ::\ :} \\
v = & \texttt{Ag-GAtT}
\end{array}$$

sera :

$$\Phi = id_1 \circ sub_{2,\texttt{G}} \circ del_3 \circ id_4 \circ id_5 \circ ins_{5,\texttt{T}} \circ id_6$$

Le poids du script sera :

$$\Omega(\Phi) = 0+1+2+0+0+2+0 = 5$$

qui correspond bien à l'opposé de la valeur du score de cet alignement.

4.3.2 Le système générateur INVERSION

De la même manière, il est possible de définir un système générateur de transformation permettant le calcul du nombre minimum d'inversions entre deux génomes (section 2.1.7, [KS93]).

Le système générateur sera défini par :

$$\mathcal{G}_{\text{INVERSION}} = \{inv\}$$

où

$$\begin{array}{rccc} inv & : & \mathcal{N}^* \times N \times N & \longrightarrow & \mathcal{N}^* \\ & & (s,p,q) & \longmapsto & s[1..p].s[q].s[q-1]\ldots s[p].s[q+1..|s|] \end{array}$$

dont le coût sera :

$$\omega(inv(s,p,q)) = 1$$

Exemple 4.8 *Soit* $S = 15324$ *et* $T = 12345$, *un script de transformation de* S *en* T *peut être :*

$$inv_{2,5} \circ inv_{3,5} \circ inv_{3,4}$$

Les étapes successives de la transformation seront :

$$15\boxed{3\ 2}4 \longrightarrow 15\boxed{2\ 3\ 4} \longrightarrow 1\boxed{5\ 4\ 3\ 2} \longrightarrow 12345$$

4.4 Le système générateur de reconstruction $\{ins,cop\}$

Nous désirons créer une mesure qui rende compte simplement de la ressemblance entre deux séquences. Nous proposons ici de reconstruire la séquence cible à partir de deux opérateurs simples : l'un exprimera les parties communes entre S et T tandis que l'autre complétera la reconstruction de T en permettant de créer directement des parties de séquences. On comprend bien que de tels scripts vont permettre de mesurer la ressemblance si l'opération de copie est d'un poids moindre que l'opération d'insertion. Le schéma envisagé va ainsi proposer l'identification de parties communes entre les deux séquences comparées. Notons qu'il n'y a pas de contrainte imposée sur l'ordre de ces paires de segments communs : ils ne doivent pas nécessairement respecter l'ordre naturel des séquences. De tels scripts vont permettre d'identifier les déplacements, les duplications (un même segment de la source peut être copié plusieurs fois dans la cible), et les insertions. Le terme d'insertion doit être entendu ici comme création de segments et pas une opération d'insertion au sens biologique du terme.

Les deux opérations que nous considérons sont donc :

1. une opération de copie (*cop*) qui permet de copier un segment de la séquence source,
2. une opération d'insertion (*ins*) qui permet de créer un segment.

En se basant sur la notion de description de longueur minimale (voir section 3.2.3), nous proposons un schéma de codage permettant de définir les fonctions de poids associées aux deux opérateurs. Le poids associé à chaque opération dépend de la quantité d'informations de ces opérations. Autrement dit, il dépend du nombre de bits nécessaires pour coder cette opération. Les opérations de copie auront ainsi un coût faible tandis que les opérations d'insertion auront un coût élevé. En effet, une copie n'est qu'une référence à une partie de la séquence source alors que l'insertion nécessite de donner explicitement, c'est-à-dire en toutes lettres, le segment à ajouter. Ainsi, plus il y aura de parties partagées par les séquences source et cible et plus le poids du script de reconstruction sera faible.

4.4.1 Les opérateurs

Le système générateur sur lequel vont être construits les scripts de reconstruction contient donc deux opérateurs :

- *ins* permet l'adjonction d'un segment quelconque v, qui ne dépend pas de la variable u

$$\begin{array}{rccc} ins & : & \Sigma^* \times \Sigma^* & \longrightarrow & \Sigma^* \\ & & (u,v) & \longmapsto & v \end{array}$$

- *cop* permet d'ajouter un segment de la séquence source u débutant en position $\lambda + p$ et de longueur l

$$\begin{array}{rccc} cop & : & \Sigma^* \times Z \times N & \longrightarrow & \Sigma^* \\ & & (u,p,l) & \longmapsto & u[\lambda + p,l] \end{array}$$

  où λ est la longueur de la séquence courante.

Cet ensemble représente un système générateur minimal permettant la reconstruction et la comparaison d'une séquence cible à partir d'une séquence source. Comme nous l'avions mentionné précédemment, il est nécessaire de disposer d'au moins un opérateur capable de créer des segments *ex-nihilo*. L'opérateur *ins* prend en charge ce travail. De plus, si nous voulons comparer les séquences, nous devons disposer d'un opérateur effectuant un traitement différent qui mette en avant la propriété de ressemblance que nous désirons exhiber. Dans le cas de l'opérateur *cop*, cette propriété est simplement le partage d'un segment. Nous reviendrons plus tard sur la notion de partage d'un segment. Pour le moment, nous considérons que partager un segment c'est avoir exactement le même segment présent dans la source et dans la cible.

La figure 4.1 représente un exemple de script de reconstruction avec le système générateur {*ins*,*cop*}.

Définition 4.5 *On appelle* ***facteur*** *une paire de segments, un segment pris dans chaque séquence (la source et la cible). Un facteur f sera notée $\langle f^s, f^t \rangle$ où f^s est un segment de S et f^t un segment de T. On notera f_p^s et f_l^s (respectivement f_p^t et f_l^t) la position et la longueur du segment f^s (respectivement f^t).*

Il est clair que l'opérateur *cop* agit sur des facteurs. Copier un segment de la séquence source dans la séquence cible correspond bien à l'opération de copie d'un facteur. Au facteur $\langle s[0..10], t[5..15] \rangle$ on associera l'opération $cop_{-5,10}(S)$. Par conséquent nous utiliserons également la notation $cop(f)$ pour désigner la copie d'un facteur f, ce qui est équivalent à $cop(S, f_p^s - f_p^t, f_l^t)$.

Nous utiliserons également $ins(f)$ pour désigner l'insertion du segment correspondant au segment couvert par f dans la séquence cible, $ins(f)$ est équivalent à $ins(S, f^t)$; ou $ins(f,g)$ pour désigner l'insertion du segment entre la fin de f et le début de g dans la séquence cible, soit à l'insertion du segment de T compris entre les positions $f_p^t + f_l^t$ et g_p^t.

4.4.2 Schéma de codage

Nous allons maintenant discuter des poids associés aux deux opérateurs *ins* et *cop*. Analysons les attributs de chaque opérateur.

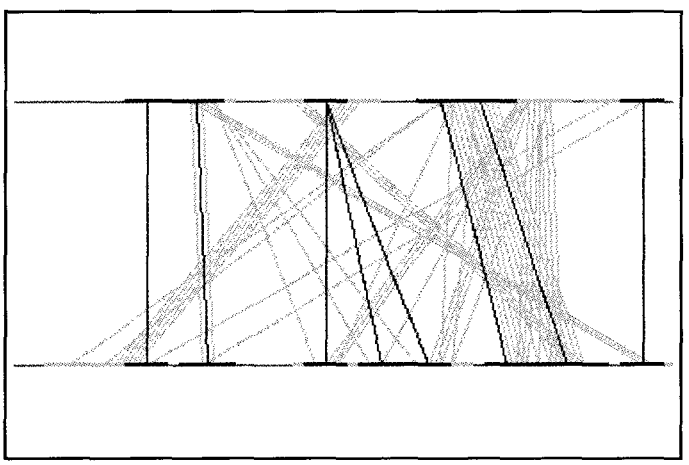
Une copie nécessite de donner la position dans la séquence source à laquelle débute le segment que nous désirons copier et la longueur de ce segment. Il s'agit donc ici de coder des positions,

Les deux séquences:

```
          1         10         20         30         40         50
S         AGAGGTAGGCTTTTTATTACACAAGTTAACTTGACATAAAAGTTAAAAGGCTTTTATAT
                  [ 1][ 2 ]        [ 3]       [ 6]                [ 8]
                                   [ 4]         [  7  ]
                                   [ 5]

T         AGATAATAACTTTCTTATTTCTGAATTAAATTAATTAAAGAATAAAAAAGTTCTTATTT
                  [ 1] [ 2 ]       [ 3] [ 4][ 5]   [ 6][  7  ]  [ 8]
```

8 facteurs ont été sélectionnés parmi les 62 facteurs de longueur supérieure ou égale à 4 qui sont communs à ces deux séquences.
Une représentation graphique est donnée ci-dessous, les traits foncés représentent les 8 facteurs sélectionnés, les traits clairs l'ensemble des 62 facteurs.



La liste des opérations:

Référence	Opération	Segment	Position courante	Position du facteur S	T
	ins(AGATAATAA)		1		
1	*cop*(0,4)	CTTT	10	10	10
	ins(C)		14		
2	*cop*(-1,5)	TTATT	15	14	15
	ins(TCTGAA)		20		
3	*cop*(0,4)	TTAA	26	26	26
	ins(A)		30		
4	*cop*(-5,4)	TTAA	31	26	31
5	*cop*(-9,4)	TTAA	35	26	35
	ins(AGA)		39		
6	*cop*(-6,4)	ATAA	42	36	42
7	*cop*(-8,7)	AAAAGTT	46	38	46
	ins(CT)		53		
8	*cop*(0,4)	TATT	55	55	55
	ins(T)		59		

FIG. 4.1 – *Exemple de script: les deux séquences sont représentées en haut et la suite des opérations est donnée en dessous.*

soit des entiers. Un entier n nécessite $\lceil \log_2(n+1) \rceil$ bits pour être codé en binaire. La position p est relative et peut donc être négative. Il faut donc ajouter un bit pour coder le signe. Si nous effectuons une copie à la position p et de longueur l, le codage correspondant à cette copie sera $1 + \lceil \log_2(|p|+1) \rceil + \lceil \log_2(l+1) \rceil$.

Une insertion nécessite de donner explicitement le segment qui doit être inséré. Cela implique qu'il faut donner le segment base par base. Chaque base pouvant être codée sur 2 bits, un segment de longueur l pourra être codé sur $2 \times l$ bits.

Nous associons donc à chaque classe d'opérateur une fonction de **poids** ω qui correspond à la longueur du codage de cet opérateur dans le langage binaire. Nous proposons les fonctions suivantes. Elles sont données à titre d'exemple et une discussion plus approfondie sur la manière de définir les poids des deux opérations *ins* et *cop* est faite section 6.2.1.

$$\omega(cop(f)) = 1 + \lceil \log_2(|p|+1) \rceil + \lceil \log_2(l+1) \rceil$$
$$\omega(ins(f)) = 2 \times l$$

Pour un propos plus clair, nous utiliserons la notation $c(f)$ pour désigner $\omega(cop(f))$ et $i(f)$ pour désigner $\omega(ins(f))$.

Des mutations dans les facteurs Le schéma de codage que nous venons de décrire correspond au cas de facteurs exacts, c'est-à-dire dont les segments dans la source et la cible sont rigoureusement les mêmes. Intéressons nous au cas de facteurs mutés dans lesquels nous autorisons des mutations ponctuelles.

Dans un premier temps, nous n'autorisons que les substitutions. Cela signifie que les deux segments ont la même longueur. Nous pouvons alors conserver le même schéma de codage, en ajoutant le codage des substitutions. Supposons avoir n substitutions aux positions $p_1, p_2, \ldots, p_n$ et notons p_0 le début du segment. Chaque substitution peut être indiquée par sa position par rapport à la substitution précédente et la substitution en elle-même peut être codée sur 2 bits. On aboutit ainsi à :

$$c(f) = 1 + \lceil \log_2(|p|+1) \rceil + \lceil \log_2(l+1) \rceil + \sum_{i=1}^{n} (\lceil \log_2(p_i - p_{i-1} + 1) \rceil + 2)$$

Nous pourrions être plus précis et coder la substitution elle-même par 1 bit si la substitution est une transition (`A` devient `G` et inversement ou `C` devient `T` et inversement) et par 2 bits si c'est une transversion (les autres subsitutions). Avec un tel codage, nous avantagons les transitions par rapport aux transversions.

Autorisons maintenant les insertions/délétions en plus des substitutions. Nous pouvons reprendre le même codage pour les positions mais cette fois il faudra être plus précis pour coder la mutation, qui peut alors être de trois types. Il est possible de prendre en compte la différence entre une insertion/délétion et une série d'insertions/délétions en codant une seule position. Dans ce cas, le codage d'une insertion de 10 bases se résumera à coder la position de début de l'insertion, sa longueur et la séquence à insérer.

Nous nous rendons compte que le codage des mutations sera fortement pénalisant et que la probabilité de recopie d'un segment muté sera fonction de la longueur du segment et du nombre

de mutations. Plus le segment sera court, plus le nombre de mutations devra être réduit pour que le segment apporte un gain de compression.

Différents codages. Le schéma de codage que nous proposons n'est pas unique. Chacun pourra proposer un codage différent relativement à l'importance qu'il veut donner à chaque opération : un schéma de codage différent fait que le modèle appréhendera différemment la notion de similarité, comme nous l'avons déjà expliqué précédemment. Par exemple il est possible de ne pas tenir compte de la différence de position, dans la source et dans la cible, des deux segments d'un facteur en utilisant un format fixe pour coder cette position. Remarquons que ce codage revient à ne pas tenir compte des positions différentes qu'il peut y avoir entre des facteurs copiés, seule la longueur intervient dans le choix d'une copie puisque toutes les positions sont codées de la même manière. Cependant, le fait de prévoir un code pour la position, même si sa longueur n'est pas variable, ne revient pas à ne pas le coder. En effet, cela joue sur le choix entre copie et insertion : on choisira plutôt de faire une insertion lorsque l'on code la position que lorsqu'on ne la code pas.

Compression globalement conservative. Nous avons vu au chapitre précédent que l'expression d'une séquence cible T au travers d'un script pouvait être interprétée comme une version comprimée de celle-ci. Nous avons discuté dans la section 3.3 des méthodes de compression non conservatives, qui font que l'on n'a pas l'assurance de pouvoir retrouver le texte initial à partir de la version comprimée de ce texte. Le schéma de codage pour lequel nous avons opté entre dans cette catégorie. En effet, si nous mettons bout à bout les codages des opérations d'un script, il sera impossible de déterminer les positions où débute et où se termine le codage d'un opérateur. Cela provient du fait que nous n'avons pas introduit d'auto-délimitation dans notre schéma de codage. Si nous donnons brutalement la séquence de bits 10110110000 en précisant que c'est un script tel qu'il est défini ici, il est impossible de dire précisément de quel script il s'agit. Les deux interprétations suivantes montrent bien l'impossibilité d'une traduction unique:

- 10 ; 1,101, 10000 $\longrightarrow$ *ins*(C),*cop*(-5,16)
- 10, 11; 0, 1, 100; 00 $\longrightarrow$ *ins*(CG),*cop*(1,4),*ins*(A)

Pourtant le fait de ne pas considérer des schémas de codage auto-délimités ne nous empêche pas d'atteindre le but que nous nous sommes fixé. Même si les codes utilisés ne nous permettent pas de reconstruire effectivement la séquence cible à partir du script et de la séquence source, nous disposons tout de même d'une mesure de la ressemblance entre les séquences. Elle est simplement définie de manière différente et s'éloigne un peu du point de vue informationnel strict qui voudrait que le script mesure exactement la quantité d'information à ajouter à S pour obtenir T.

4.4.3 Distance par copies de facteurs

La mesure de distance que nous mettons en place est simplement l'application de la mesure générale définie dans la section précédente :

$$\mathrm{d}_C = \min_{\{\varphi \mid \varphi(S)=T\}} \omega(\varphi)$$

4.5 Résumé

Nous avons proposé dans ce chapitre la définition de la **distance de transformation** comme la longueur du **script de transformation** le plus court permettant de transformer une séquence source S en une séquence cible T. Cette notion définit une famille de distances, chacune d'entre-elles étant une spécification par le choix d'un système générateur particulier et de fonctions de poids associées.

Nous nous sommes plus particulièrement intéressés au système générateur de reconstruction {ins,cop} permettant de reconstruire une séquence cible T à partir de segments pris dans une séquence source S. Nous avons montré que la mesure de la quantité d'information d'un script minimal construisant T à partir de S permet de quantifier la ressemblance entre S et T. La donnée du script minimal, c'est-à-dire de la suite d'instructions, permet quant à elle de qualifier la ressemblance entre S et T.

Le chapitre suivant traite d'algorithmes pour le calcul de ces scripts de reconstruction dans le système générateur $\{ins,cop\}$. Un algorithme exhaustif est d'abord présenté puis deux algorithmes dont l'exactitude dépend de conditions sur les fonctions de poids. Le chapitre 6 spécifie un peu plus la distance avec des choix sur les fonctions de poids. On y examine également les propriétés de la distance par copies de facteurs et on y fait une comparaison des résultats obtenus par les différents algorithmes.

Chapitre 5

Graphe de scripts

Nous montrons comment le problème de la recherche d'un script optimal basé sur le système générateur de reconstruction $\{ins,cop\}$ peut être résolu en le transformant en un problème de recherche de plus court chemin dans un graphe.

La première partie de ce chapitre considère le cas général et propose un algorithme permettant de calculer la distance par la recherche du plus court chemin dans un graphe acyclique, orienté et pondéré. Nous montrons dans un premier temps qu'il y a équivalence entre la donnée d'un script et la donnée d'un ensemble de facteurs non chevauchants. Il est en effet possible de dériver un script de reconstruction avec les opérations de copie et d'insertion à partir d'un ensemble de facteurs non chevauchants et communs à S et T. Ainsi la recherche d'un script minimum équivaut à la recherche d'une combinaison de facteurs dont le script associé est de poids minimum. Il est alors possible de construire un graphe où chaque sommet va représenter une possible opération de copie et chaque arc l'opération d'insertion nécessaire entre deux opérations de copie successives. Il suffit alors de calculer le chemin le plus court dans ce graphe, en définissant les poids des arcs de manière adéquate, pour calculer le script minimum. Nous proposons un algorithme utilisant des résultats connus sur les recherches de chemin les plus courts dans des graphes orientés, acycliques, pondérés et à source unique.

La recherche d'un script minimum avec cette technique est gourmande en temps et en mémoire si l'ensemble des facteurs communs est grand ou s'il existe de nombreux facteurs qui se chevauchent. Les deux sections suivantes du chapitre identifient des conditions sur les fonctions de poids i et c qui, si elles sont vérifiées, permettent de diminuer la taille du graphe, tant par le nombre de facteurs que par le nombre d'arcs. Lorsque ces conditions sont satisfaites, nous disposons d'un algorithme exact et efficace du calcul du script minimal. Si ces conditions ne sont pas vérifiées, les algorithmes proposés sont heuristiques. Ils permettent un calcul approché qui en pratique fourni des résultats satisfaisants la plupart du temps. La comparaison des résultats des différents algorithmes sera étudiée dans le chapitre suivant.

Dans un premier temps, nous mettons en avant des propriétés de non-optimalité de certains scripts. Ces propriétés sont conditionnées par des relations que les fonctions de poids des deux opérateurs *ins* et *cop* doivent vérifier. Bien souvent ce sont des propriétés de bon sens que les fonctions de poids des opérateurs vérifient naturellement si elles sont définies pour avantager les opérations de copies. Ces propriétés permettent d'améliorer la rapidité de calcul de la distance, sans pour autant changer la classe de complexité. Cependant, la pratique montre que le gain

est tout à fait significatif. Dans un second temps, nous nous intéressons au cas particulier où l'ensemble des facteurs contient des facteurs qui sont *a priori* toujours valables. C'est-à-dire que leur présence dans un script permet toujours d'obtenir un script de poids plus faible que leur abscence.

La technicité de ce chapitre peut rendre sa lecture assez difficile. Nous nous sommes attachés à agrémenter notre propos de nombreux schémas permettants une meilleure appréhension des concepts que nous exposons. En particulier, nous utilisons les séquences et les facteurs de l'exemple 5.1 pour illustrer les définitions des trois algorithmes présentés. Le lecteur pourra ainsi apprécier les améliorations et modifications apportées.

5.1 Le problème vu sous forme de graphe

Nous définissons dans cette section une structure de graphe permettant de représenter le problème du calcul de la distance de transformation basée sur le système générateur de reconstruction $\{ins, cop\}$. Nous allons voir que le problème de la recherche d'un script de poids minimum équivaut à la recherche d'un chemin le plus court dans un graphe.

Dans un premier temps, nous définissons la notion de $f-$script qui est une représentation d'un script comme l'ensemble des facteurs qui interviennent dans les opérations de copie. Nous montrons que l'ensemble des $f-$script peut alors être modélisé sous la forme d'un graphe orienté, acyclique. En affectant des poids aux arêtes de ce graphe, il y a alors équivalence entre le poids d'un chemin dans le graphe (qui est la somme des poids des arcs de ce chemin) et le poids du $f-$script qui lui est associé. Nous proposons alors une méthode permettant de résoudre le problème de la recherche d'un script optimal en temps polynomial.

5.1.1 Notion de $f-$script

Préliminaires

Nous définissons deux relations sur les facteurs. La première d'entre-elles définit un ordre sur les facteurs. La seconde permet d'identifier des facteurs qui ne se chevauchent pas dans la séquence cible.

Définition 5.1 *On dira que f* ***précède*** *g, et on notera $f <_s g$ si f débute avant g dans la séquence cible ou si les deux facteurs débutent à la même position dans la séquence cible et f est plus court que g (l'indice s indique le rapport avec l'ordre sur les séquences).*

Définition 5.2 *On dira que f* ***précède strictement*** *g, et on notera $f <_o g$, si $f <_s g$ et si f se termine avant le début de g dans la séquence cible (l'indice o indique la relation avec le chevauchement, overlap en anglais).*

Nous donnons la définition d'une troisième relation indiquant que deux facteurs se chevauchent.

Définition 5.3 *On dira que f* ***chevauche*** *g, et on notera $f \not<_o g$, si $f <_s g$ et si f et g ne vérifient pas la relation $<_o$.*

La relation $<_s$ est une relation d'ordre strict sur l'ensemble des facteurs. En effet $<_s$ est transitive (si $f <_s g$ et $g <_s h$ alors $f <_s h$) et pour tout couple de facteurs (f,g) tel que $f \neq g$, on a nécessairement $f <_s g$ ou $g <_s f$. La relation $<_o$ est une relation d'ordre partielle. Celle-ci est transitive mais deux facteurs ne sont par forcément en relation par rapport à $<_o$: deux facteurs qui se chevauchent ne vérifient pas la relation $<_o$.

L'ensemble des facteurs communs à deux séquences peut être ordonné suivant la relation $<_s$. Les facteurs apparaissent alors dans l'ordre de leur position de début dans la séquence cible T (la séquence source sera notée S), et, pour une même position, dans l'ordre croissant de leur longueur.

Nous introduisons une notation permettant de désigner les ensembles de facteurs qui vérifient la relation $<_o$ avec un autre facteur.

Définition 5.4 *Soit $\varphi \subset F$ un ensemble de facteurs,*

$pred(\varphi) = \{f \in F \mid \exists\, g \in \varphi,\ f <_o g\}$ désigne l'ensemble des ***prédécesseurs*** *de φ,*

$succ(\varphi) = \{f \in F \mid \exists\, g \in \varphi,\ g <_o f\}$ désigne l'ensemble des ***successeurs*** *de φ.*

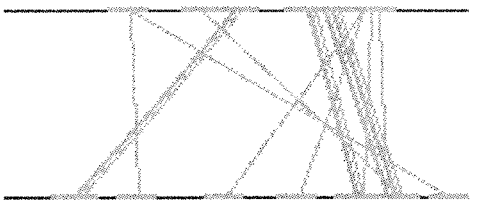
Si φ ne comporte qu'un seul facteur f, on notera $pred(f)$ et $succ(f)$ respectivement les prédécesseurs et successeurs de f.

Dans la suite, nous représenterons les facteurs communs aux deux séquences uniquement par le segment qu'ils couvrent dans la séquence cible. **Si un facteur précède un autre alors le premier sera dessiné au dessus du second.** Une illustration des relations définies ci-dessus et du mode de représentation que nous avons adopté est donnée dans l'exemple ci-dessous.

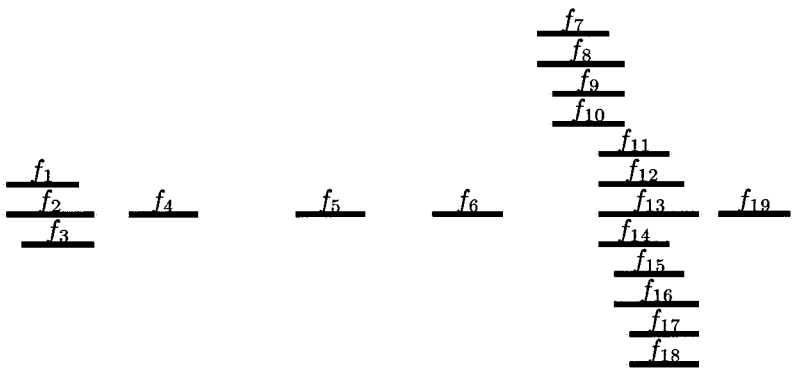
Exemple 5.1 *Considérons l'ensemble des facteurs suivant entre les deux séquences S et T de l'exemple 5.1 (seuls les facteurs de taille supérieure ou égale à 5 sont sélectionnés). Un triplet (p,q,l) représente un facteur où p est la position du facteur dans S, q la position du facteur dans T et l sa longueur.*

1 = 27 7 5 *2 = 27 7 6*
3 = 28 8 5
4 =14 15 5
5 =43 26 5
6 =43 35 5
7 =36 42 5 *8 =36 42 6*
9 =37 43 5 *10=44 43 5*
11=38 46 5 *12=38 46 6* *13=38 46 7* *14=45 46 5*
15=39 47 5 *16=39 47 6*
17=23 48 5 *18=40 48 5*
19=14 54 5

Le schéma ci-dessous représente les facteurs sur les deux séquences :



La représentation que nous adoptons donne le schéma suivant :



Seule le segment correspondant à la séquence cible est représentée pour chaque facteur. On a ici $f_1 <_s f_2 <_s f_3 <_s \ldots <_s f_{19}$. *On vérifie qu'entre autres* $f_1 <_o f_4$ *et* $f_2 <_o f_4$ *et* $f_1 \not<_o f_2$.

Définition d'un $f-$script

Lorsque nous cherchons un script de description de la séquence cible, nous cherchons quelles parties de la séquence cible doivent être copiées à partir de la séquence source et quelles parties doivent être décrites explicitement. En fait, si nous disposons uniquement des parties qui doivent être copiées alors le script est entièrement défini. En effet, lors du calcul du script de reconstruction, nous disposons des deux séquences. Par conséquent, si nous connaissons tous les facteurs mis en jeu dans les opérations de copie d'un script, nous pouvons déduire les parties qui doivent être créées par complémentarité pour aboutir à la séquence cible.

Un algorithme de recherche d'un script équivaut à rechercher une combinaison de facteurs.

Nous introduisons la notion de $f-$**script** afin de formaliser cette idée. Un $f-$script va représenter un ensemble de facteurs qui ne se chevauchent pas dans la séquence cible. La propriété de non-chevauchement est une conséquence du fait que nous nous intéressons ici à une reconstruction. Par conséquent, nous nous interdisons de construire deux fois la même portion de la

séquence cible.

Définition 5.5 *Une* ***liste de facteurs*** $(f_1,f_2,\ldots,f_n)$ *est un ensemble de facteurs ordonnés par la relation* $<_s$.

Définition 5.6 *Soit F un ensemble de facteurs entre deux séquences S et T, un* $f-$**script** S_f *est une liste de facteurs de F telle que pour tout couple (f,g) de S_f, si $f <_s g$ alors $f <_o g$.*

Il est facile de déduire d'un $f-$script un script de reconstruction de la séquence cible T. En effet, un $f-$script $(f_1,f_2,\ldots,f_n)$ est la donnée d'un ensemble de segments de la séquence cible. Il existe toujours une liste de segments de T $(t_0,\ldots,t_n)$ telle que:

$$T = t_0.f_1^t.t_1.f_1^t\ldots f_n^t.t_n$$

Par conséquent, le script de reconstruction effectuant les insertions des t_i et les copies des f_j^t permet la reconstruction de S à partir de T.

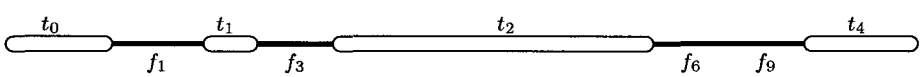
Proposition 5.1 *Soit $\phi = (f_1,f_2,\ldots,f_n)$ un $f-$script entre S et T, soient $(t_0,\ldots,t_n)$ les segments de T tels que $T = t_0.f_1^t.t_1.f_1^t\ldots f_n^t.t_n$, $R = (ins_{t_0},cop_{f_1},ins_{t_1},\ldots,cop_{f_n},ins_{t_n})$ est un script de reconstruction tel que $R(S) = T$.*

Preuve 5.1

Le calcul de $R(S)$ permet de trouver la séquence $t_0.f_1^t.t_1.f_1^t\ldots f_n^t.t_n$, c'est-à-dire T. En effet $R(S)$ est la concaténation des résultats de chaque opérateur de reconstruction, c'est-à-dire $ins_{t_0}(S).cop_{f_1}(S).ins_{t_1}(S)\ldots cop_{f_n}(S).ins_{t_n}(S)$ □

Exemple 5.2 *Dans l'exemple 5.1, un $f-$script pourra être $\{f_1,f_3,f_6,f_9\}$, la reconstruction associée sera alors:*

$$ins_{t_0},cop_{f_1^t},ins_{t_1},cop_{f_3^t},ins_{t_2},cop_{f_6^t},cop_{f_9^t},ins_{t_4}$$



Dans le cas présent, il n'y a pas d'insertion d'un segment t_3 car f_6 se termine là où débute f_9 (on peut aussi considérer que l'on insère un segment vide, dont le coût est nul).

Poids d'un $f-$script

Nous définissons le poids d'un $f-$script comme le poids de la reconstruction qui lui est associée. Il est nécessaire pour cela de considérer deux facteurs supplémentaires. L'un est un facteur initial 'a', permettant de mesurer le coût d'insertion nécessaire entre le début de la séquence cible (c'est-à-dire la position 0) et le premier facteur du $f-$script. L'autre est un facteur terminal et permet de mesurer le coût d'une insertion entre le dernier facteur du $f-$script et la fin de la séquence 'z'.

Définition 5.7 *Le poids d'un $f-$script $\phi = (f_1,f_2,\ldots,f_n)$ est:*

$$w(\phi) = i(a,f_1) + \sum_{i=1}^{n} c(f_i) + \sum_{i=1}^{n-1} i(f_i,f_{i+1}) + i(f_n,z)$$

avec $a = \langle 0,0,0\rangle$ et $z = \langle |S|,|T|,0\rangle$.

Ainsi, la recherche d'un script de reconstruction bâti sur le système générateur {ins,cop}, permettant d'obtenir T à partir de S, et de poids minimal, correspond à la recherche d'un f−script entre S et T de poids minimal.

Proposition 5.2 *Soient S et T deux séquences, il existe une application bijective qui associe un f−script entre S et T à chaque script bâti sur le système générateur {ins,cop}.*

Preuve 5.2

Soit $\phi = (f_1,f_2,\ldots,f_n)$ un $f-$script entre S et T, la proposition 5.2 montre que ϕ équivaut au script $R = (ins_{t_0},cop_{f_1},ins_{t_1},\ldots,cop_{f_n},ins_{t_n})$. Supposons qu'il existe un autre script R' différent de R et équivalent à ϕ. Il est clair que $f_1^t f_2^t \ldots f_n^t$ est une sous-séquence de T et donc R' doit contenir les opérations $cop_{f_1},cop_{f_2},\ldots,cop_{f_n}$ dans cet ordre. R' ne peut contenir d'autres opérations de copie (sinon des opérations de copie de longueur nulle dont nous ne nous préoccupons pas). Les opérations d'insertion sont nécessairement les mêmes que celles de R pour pouvoir obtenir la séquence T. La démonstration qu'il n'existe qu'un $f-$script possible équivalent à un script donné est basée sur le même argumentaire. □

5.1.2 Définition du graphe de scripts

Nous présentons maintenant la notion de **graphe de scripts** qui permet une représentation sous forme d'un graphe de l'ensemble des scripts possibles pour reconstruire la séquence cible à partir de la séquence source. Chaque script est représenté par le $f-$script qui lui correspond. Le graphe va être composé d'un ensemble de sommets, représentant les facteurs, et d'un ensemble d'arcs représentant les successions possibles de deux facteurs dans un script. Chaque chemin dans le graphe va représenter un $f-$script, c'est-à-dire une combinaison de facteurs. En associant aux arcs du graphe des poids appropriés, le poids d'un chemin (égal à la somme des poids des arcs) représentera le poids du $f-$script.

Chaque sommet du graphe va représenter une opération possible de copie. On ajoute deux sommets virtuels a et z représentant la source et le but du graphe auxquels sont associés respectivement les deux facteurs virtuels a et z que nous avons rencontrés plus haut. Un arc va relier deux sommets i et j si les facteurs représentés par ces sommets, respectivement f et g, sont tels que $f <_o g$. Il existe donc un arc entre a et tous les sommets du graphe, et un arc entre tous les sommets du graphe et z. On associe aux arcs un poids qui va correspondre à la somme du poids de l'insertion nécessaire entre les deux sommets du graphe et du poids nécessaire à la copie du sommet origine de l'arc.

Définition 5.8 *Un **graphe de scripts** G est un triplet (V,E,w) où*

- $V = F \cup \{a,z\}$ *est l'ensemble des sommets,*
- $E = \{(f,g) \mid f,g \in V, f <_o g\}$ *est l'ensemble des arcs,*
- w *est une fonction qui associe un poids à chaque arc : pour tout arc* (f,g) *de* E, $w((f,g)) = c(f) + i(f,g)$.

Dans la suite, on parlera indifféremment de facteur ou de sommet pour désigner le sommet ou le facteur représenté par le sommet d'un graphe. Suivant le contexte, f désignera soit un sommet, soit le facteur qui lui est associé.

Un exemple de graphe de scripts basé sur l'ensemble des facteurs de l'exemple 5.1 est donné figure 5.1.

Proposition 5.3 *Un chemin de a à z de coût minimum dans un graphe de scripts $G = (V,E,w)$ construit sur en ensemble de facteurs entres deux séquences S et T correspond à un $f-$script de poids minimum.*

Preuve 5.3

Nous montrons successivement qu'un chemin de a à z correspond à un $f-$script, que tous les $f-$scripts sont dans le graphe et que le poids d'un chemin est égal au poids du $f-$script qui lui est associé :

- Un chemin de a à z dans un graphe de scripts $G = (V,E,w)$ représente un $f-$script. Soit $a,f_1,f_2,\ldots,f_n,z$ un chemin dans le graphe. Notons a, f_0 et z, f_{n+1}. Pour tout i appartenant à $0..n$, (f_i,f_{i+1}) est un arc de E et donc $f_i <_o f_{i+1}$. L'ensemble $\{f_1,f_2,\ldots,f_n\}$ est donc un sous-ensemble des facteurs entre S et T tel que pour tous facteurs f,g on a $f <_o g$ ou $g <_o f$. Cela correspond exactement à la définition de $f-$script.
- Tous les $f-$scripts permettant la reconstruction de T à partir de S sont dans le graphe de scripts construit sur l'ensemble des facteurs entre S et T. La preuve est évidente à partir de la définition du graphe de scripts. Tous les facteurs sont dans le graphe et tous les facteurs respectant la relation $<_o$ sont reliés par un arc.
- Le coût d'un chemin dans le graphe est égal au coût du $f-$script associé. Le coût d'un chemin dans un graphe est la somme des coûts de chaque arête. Ainsi, le coût d'un chemin $f_0,f_1,f_2,\ldots,f_n,f_{n+1}$ est $\sum_{i=0}^{n} w((f_i,f_i+1)) = \sum_{i=0}^{n}(c(f_i)+i(f_i,f_i+1))$.

□

La recherche d'un $f-$script de poids minimum correspond à la recherche d'un chemin de poids minimum reliant a à z. Un chemin de a à z donne une combinaison de facteurs entre S et T, et par conséquent un $f-$script. On peut donc en déduire un script. Le poids d'un chemin du graphe est la somme des poids des arcs de ce chemin. La définition du poids d'un arc fait que le poids d'un chemin correspond bien au poids du $f-$script associé et donc au poids du script associé.

5.1.3 Résolution

Un tel graphe de scripts nous permet de résoudre le problème de la recherche d'un script minimum en appliquant un algorithme de recherche du plus court chemin dans un graphe orienté, acyclique, pondéré à source unique. Plus particulièrement, nous allons utiliser l'algorithme PLUS-COURTS-CHEMINS-GOA (recherche d'un plus court chemin dans un graphe orienté acyclique) décrit dans [CLR94]. On affecte un poids à chaque sommet qui correspond au poids du plus court chemin permettant d'aboutir à ce sommet. L'algorithme envisage les sommets dans l'**ordre topologique** : s'il existe un chemin joignant les sommets f et g, alors f précède g pour l'ordre topologique. Pour chaque sommet f, le poids de chaque sommet successeur g est mis à jour : le poids du sommet g est révisé suivant qu'il est plus court de passer par f pour atteindre g que de passer par l'actuel plus court chemin. Cette opération est appelé **relâchement**.

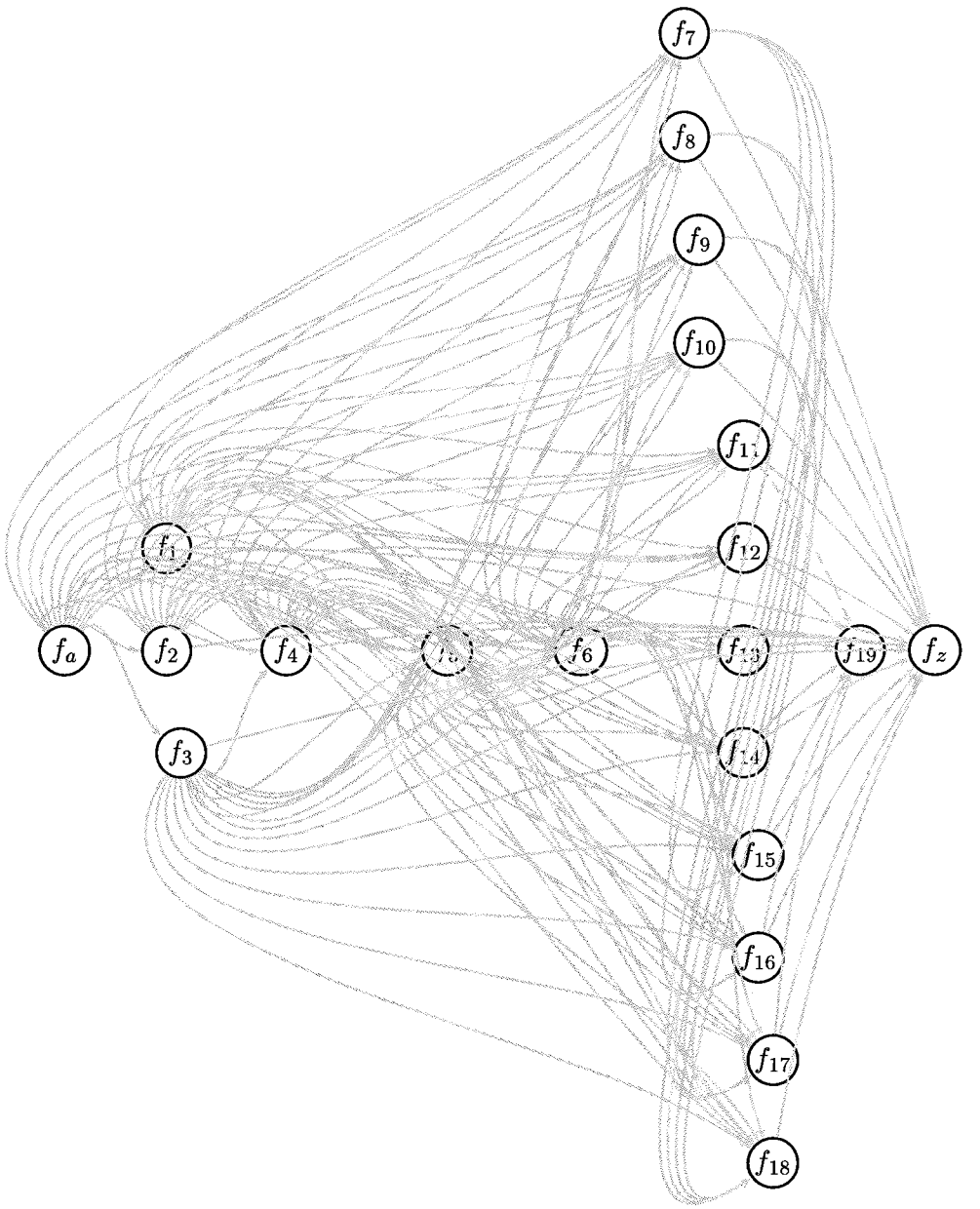


FIG. 5.1 – *Exemple de graphe de scripts pour l'ensemble de facteurs de l'exemple 5.1. Chaque facteur est relié à tous ses successeurs (pour la relation $<_o$). a est relié à $\{f_1, \ldots, f_{19}, z\}$. f_7 est relié à $\{f_{15}, f_{16}, f_{17}, f_{18}, f_{19}, f_z\}$ car $f_7 \not<_o f_{11}$, $f_7 \not<_o f_{12}$, $f_7 \not<_o f_{13}$, $f_7 \not<_o f_{14}$. Le graphe comporte 157 arcs.*

Dans le cas général, il est nécessaire de trier les sommets du graphe pour construire l'ensemble des sommets ordonné suivant l'ordre topologique. Dans le cas du graphe de scripts, cet ordre correspond exactement à l'ordre défini par la relation $<_s$. En effet, il ne peut y avoir d'arc entre deux sommets du graphe de scripts que si les facteurs associés vérifient la relation $<_s$.

Relâchement.

Ce procédé est basé sur la propriété importante que les sous-chemins des plus courts chemins sont eux-mêmes des plus courts chemins. Avec les notations du graphe de script, il est facile de montrer que pour tout arc (f,g) on vérifie $w(a,g) \leq w(a,f)+w(f,g)$. Il est donc facile de disposer d'une variable, qui, pour chaque sommet, indique le poids minimum d'un chemin joignant a et ce sommet. Le procédé de relâchement d'un arc (f,g) consiste donc en un test déterminant s'il est possible d'obtenir un chemin plus court pour aboutir à g en passant par f. La procédure suivante présente le relâchement de l'arc (f,g).

Notation 5.1 *$w(f)$ désigne le poids associé au sommet f.*

ALG 5.1 RELÂCHER

si $w(g) > w(f) + w(f,g)$ **alors**
  $w(g) \leftarrow w(f) + w(f,g)$
fin si

Pour que la technique du relâchement fonctionne, il faut avant cela initialiser les poids des sommets du graphe. Le poids de a est initialisé à zéro, les poids des autres sommets le sont à $+\infty$.

Notation 5.2 *$\delta(f,g)$ désigne le poids du plus court chemin joignant les sommets f et g. Pour tout f, $\delta(f,f) = 0$ par convention.*

Lemme 5.1 ([CLR94]) *Soit $G = (V,E,w)$ un graphe de scripts. Alors $w(f) \geq \delta(a,f)$ pour tout sommet f de V. Cet invariant est conservé au cours de toutes les étapes de relâchement des arcs de G. De plus, une fois que $w(f)$ a atteint sa borne inférieure $\delta(a,f)$, il n'est plus jamais modifié.*

Preuve 5.1

Après initialisation, on a effectivement $w(f) \geq \delta(a,f)$ puisque $w(a) = 0 \geq \delta(a,a)$ et $w(f) = +\infty \geq \delta(a,f)$ pour tout f différent de a. Supposons qu'il existe un sommet g qui soit le premier sommet pour lequel un relâchement d'un arc (f,g) fasse que $w(g) < \delta(s,g)$. Après le relâchement, on aura :

$$\begin{aligned} w(f) + w(f,g) &= w(g) \\ &\leq \delta(a,g) \\ &\leq \delta(a,f) + w(f,g) \end{aligned}$$

et donc $w(f) < \delta(a,f)$. Or $w(f)$ n'est pas modifié par le relâchement de l'arc (f,g), ce qui contredit l'hypothèse que g soit le premier sommet pour lequel l'invariant n'est plus vérifié. Dès que $w(f)$ a atteint sa borne inférieure $\delta(a,f)$, il n'est plus jamais modifié puisque d'une part on a $w(f) \geq \delta(a,f)$ et d'autre part $w(f)$ n'est jamais augmenté par le relâchement d'un arc. □

Lemme 5.2 ([CLR94]) *Soit $G = (V,E,w)$ un graphe de scripts. Soient f et g deux sommets de V tels que $a,\ldots,f,g$ soit un plus court chemin de a à g. Supposons qu'il ait été effectué une série de relâchements incluant le relâchement de l'arc (f,g). Si $w(f) = \delta(a,f)$ avant ce relâchement, alors on aura $w(g) = \delta(a,g)$ après.*

Preuve 5.2

Après le relâchement de (f,g), on aura

$$\begin{array}{rcll} w(g) & \leq & w(f) + w(f,g) & \\ & = & \delta(a,f) + w(f,g) & \text{d'après le lemme 5.1} \\ & = & \delta(a,g) & \end{array}$$

D'après le lemme 5.1, on sait que $\delta(a,g)$ est une borne inférieure de $w(g)$, l'égalité est donc maintenue par la suite. □

Algorithme du plus court chemin.

L'algorithme de recherche de chemin le plus court dans un graphe orienté acyclique consiste simplement en un relâchement de tous les sommets avec ses successeurs.

ALG 5.2 PLUS-COURTS-CHEMINS-GOA

Entrée: Un graphe $G = (V,E,w)$ orienté, pondéré, acyclique à source unique

$w(a) \leftarrow 0$
pour tout sommet $f \in V - \{a\}$ **faire**
  $w(f) \leftarrow +\infty$
fin pour
pour tout sommet $f \in V$ pris dans l'ordre topologique **faire**
  pour tout sommet $g \in \{f,(f,g) \in E\}$ **faire**
    RELÂCHER(f,g)
  fin pour
fin pour

Un exemple d'exécution est donné figure 5.2.

Le temps d'exécution de l'algorithme est $\theta(V+E)$ si l'ensemble des sommets nécessite d'être trié topologiquement [CLR94, chapitre 23]. Si au contraire l'ensemble des sommets est déjà trié, alors la complexité en temps est $\theta(E)$, ce qui correspond au nombre d'appels à la procédure RELÂCHER.

Proposition 5.4 *Si $G = (V,E,w)$ est un graphe de scripts alors après l'exécution de* PLUS-COURTS-CHEMINS-GOA, *$w(f)$ est le poids minimal d'un chemin entre a et f pour tout sommet f.*

Preuve 5.4

G est un graphe acyclique, orienté, pondéré, à source unique a et tel que tout sommet est accessible à partir de a. Si f est un facteur quelconque de V alors

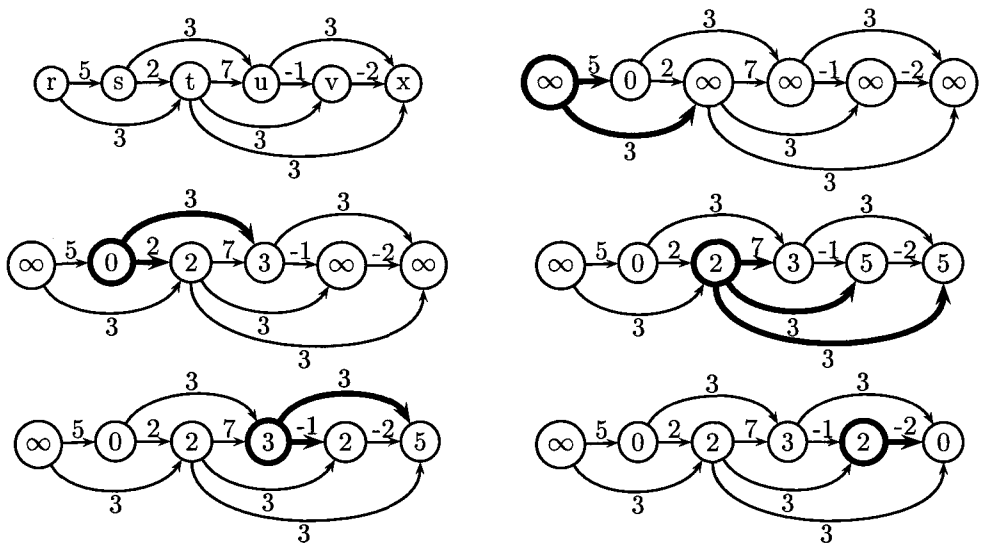


FIG. 5.2 – *Exemple d'exécution de l'algorithme* PLUS-COURTS-CHEMINS-GOA. *Les schémas sont ordonnés de gauche à droite et de haut en bas. On dispose au départ d'un ensemble de 6 sommets. Le sommet initial est s, il est initialisé à 0. Les autres sommets sont initialisés à* ∞. *On relâche ensuite les sommets dans l'ordre topologique. r est le premier sommet relâché : les poids des sommets accessibles par un seul arc à partir de r. Le poids de s reste 0 puisque* $\infty + 5 > 0$, *idem pour t. On envisage ensuite s, le poids de t devient égal à 2 et le poids de u devient égal à 3. A la fin, le poids du chemin le plus court est 0, il correspond au chemin s,u,v,x.*

il existe un plus court chemin $(a = f_0, f_1, \ldots, f_n = z)$ dans le graphe. Les sommets étants traités dans l'ordre $<_s$, les relâchements des arcs se font dans l'ordre : $(f_0,f_1),(f_1,f_2),\ldots,(f_{n-1},f_n)$. Le lemme 5.2 permet de montrer par récurrence que pour tout $i \in 0..n$ on aura que $w(f_i)$ contient le poids du chemin le plus court permettant d'aller de a à f_i. □

Calcul de la distance de transformation.

Dans l'algorithme 5.3 nous présentons une implémentation du calcul de la distance de transformation basée sur la représentation du graphe de scripts et utilisant l'algorithme PLUS-COURTS-CHEMINS-GOA pour calculer le script de poids minimum. La proposition précédente montre la correction de cet algorithme.

Remarquons que le calcul de la distance est effectué sans avoir besoin de construire effectivement le graphe.

Le tableau 5.1 est un exemple d'exécution de l'algorithme MEILLEURSCRIPT avec l'ensemble des facteurs de l'exemple 5.1.

x	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	f19	z
	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
w(a,x)	14	14	16	30	52	70	84	84	86	86	92	92	92	92	94	94	96	96	108	118
	14	14	16	30	52	70	84	84	86	86	92	92	92	92	94	94	96	96	108	118
w(f1,x)				14	36	54	68	68	70	70	76	76	76	76	78	78	80	80	92	102
				28	50	68	82	82	84	84	90	90	90	90	92	92	94	94	106	116
w(f2,x)				12	34	52	66	66	68	68	74	74	74	74	76	76	78	78	90	100
				26	48	66	80	80	82	82	88	88	88	88	90	90	92	92	104	114
w(f3,x)				12	34	52	66	66	68	68	74	74	74	74	76	76	78	78	90	100
				26	48	66	80	80	82	82	88	88	88	88	90	90	92	92	104	114
w(f4,x)					16	34	48	48	50	50	56	56	56	56	58	58	60	60	72	82
					42	60	74	74	76	76	82	82	82	82	84	84	86	86	98	108
w(f5,x)						16	30	30	32	32	38	38	38	38	40	40	42	42	54	64
						58	72	72	74	74	80	80	80	80	82	82	84	84	96	106
w(f6,x)							11	11	13	13	19	19	19	19	21	21	23	23	35	45
							69	69	71	71	77	77	77	77	79	79	81	81	93	103
w(f7,x)															6	6	8	8	20	30
															75	75	77	77	89	99
w(f8,x)																	6	6	18	28
																	75	75	87	97
w(f9,x)																	6	6	18	28
																	75	75	87	97
w(f10,x)																	4	4	16	26
																	75	75	87	97
w(f11,x)																			13	23
																			87	97
w(f12,x)																			11	21
																			87	97
w(f13,x)																			9	19
																			86	96
w(f14,x)																			10	20
																			86	96
w(f15,x)																			11	21
																			86	96
w(f16,x)																			9	19
																			84	94
w(f17,x)																			10	20
																			84	94
w(f18,x)																			9	19
																			84	94
w(f19,x)																				9
	14	14	16	26	42	58	69	69	71	71	77	77	77	77	75	75	75	75	84	93

TAB. 5.1 – *Exécution de* MEILLEURSCRIPT *pour les facteurs de l'exemple 5.1. La première ligne donne les poids initiaux des sommets. La dernière les poids après relâchement de tous les arcs. Le script de poids le plus faible a un poids de 93. La première ligne de chaque couple de lignes donne le coût des arcs. La quatrième colonne de la ligne $w(a,x)$ donne le coût de l'arc (a,f_4), égal à 30. La seconde ligne donne le poids associé à chaque sommet après l'opération de relâchement. La quatrième colonne donne le poids du sommet f_4. Ainsi, après le relâchement de (f_2,f_4), f_4 a un poids de 26 car $w(f_2) + w(f_2,f_4) = 14 + 12 < 28 = w(f_4)$.*

ALG 5.3 MEILLEURSCRIPT : Calcul de la distance de transformation.
Entrée: F un ensemble de facteurs ordonné suivant la relation $<_s$
Sortie: la distance de transformation (par copie de facteurs)

```
{ initialisation des poids }
pour tout f ∈ F ∪ {z} faire
  w(f) ← +∞
fin pour
w(a) ← 0
pour tout f ∈ F ∪ {a,z} faire
  pour tout g ∈ F ∪ {a,z} tel que f <_o g faire
    { g représente les facteurs successeurs de f qui ne le chevauchent pas }
    si w(g) > w(f) + w(f,g) alors
      { relâchement }
      w(g) ← w(f) + w(f,g)
    fin si
  fin pour
fin pour
retourner w(z)
```

Analyse de la complexité.

L'algorithme PLUSCOURTSCHEMINSGOA effectue la recherche d'un chemin minimal en un temps $\theta(E)$ si le graphe est déjà construit et trié dans l'ordre topologique. Si le nombre de facteurs identifiés entre S et T est n, alors le graphe de scripts comporte $n+2$ noeuds. Chaque noeud est relié à tous les noeuds tels que leurs facteurs respectifs vérifient la relation $<_o$. Si l'on prend les noeuds dans l'ordre $<_s$, cela veut dire que chaque noeud peut dans le pire des cas être relié à tous ses successeurs. Le premier noeud ('a') est relié aux $n+1$ suivants, le second peut-être relié aux n suivants et donc le i-ème aux $n-(i-2)$ suivants. On en déduit que le nombre d'arcs est au maximum

$$\sum_{i=1}^{n+2} n-(i-2) = \frac{n^2+3n+2}{2}$$

et par conséquent la complexité de la recherche du script minimal pour n facteurs est en $\theta(n^2)$ si l'on dispose du graphe de scripts.

Dans l'algorithme que nous donnons le graphe n'est pas construit, nous faisons comme si celui-ci l'était en cherchant à chaque étape l'ensemble des sommets g ne chevauchant pas f. Si nous devions construire le graphe, ce serait ce que nous ferions et le temps d'exécution serait en $\theta(n^2)$. S'affranchir de la construction du graphe n'augmente donc pas la complexité totale tout en préservant l'espace mémoire qui ici est limité au stockage des sommets. L'algorithme MEILLEURSCRIPT calcule donc la distance par copie de facteurs en un temps $\theta(n^2)$ et avec une complexité en espace en $\theta(n)$.

Considérons nos deux séquences S et T et supposons que la plus grande est de longueur l, alors il existe au maximum l^3 facteurs entre S et T. Par conséquent, la recherche d'un script minimum entre S et T est en $O(l^6)$ si la plus grande des deux séquences est de longueur l. Bien

sûr il existe rarement des séquences possédant un nombre de facteurs d'ordre de grandeur $O(l^3)$. Cela correspond au cas très particulier où les deux séquences sont identiques et composées d'une seule base. Cette complexité correspond au pire des cas et est évaluée de manière très large puisqu'il est impossible de calculer de façon précise le nombre de facteurs entre deux séquences (sauf à les énumérer). La complexité en temps est donc polynomiale et la complexité en espace est linéaire par rapport au nombre de facteurs et polynomiale par rapport à la longueur des séquences (en $O(l^3)$).

5.2 Graphe de scripts compact

Obtenir un algorithme dont la complexité est polynomiale paraît suffisant. Il se trouve que si l'on veut traiter des données de grande taille, le calcul de la distance de transformation par la méthode du graphe de scripts ne permet pas d'aboutir à des résultats en un temps satisfaisant. Cela dépend bien sûr du type de données mais si l'on considère deux séquences assez similaires, le nombre de facteurs communs est trop grand pour que le calcul du meilleur chemin se fasse en un temps raisonnable. Cela provient essentiellement du fait qu'il y aura de nombreux facteurs qui se chevauchent et donc une multiplication des arcs. Une autre cause est la prise en compte de tous les facteurs alors que certains ne peuvent pas aboutir à une solution optimale. Nous décrivons donc ici un algorithme diminuant la taille du graphe de scripts (en nombre de sommets et en nombre d'arcs). Cette réduction est obtenue en observant plusieurs propriétés, qui sont des inégalités sur les fonctions de poids. Si ces inégalités sont vérifiées, alors l'algorithme proposé calcule exactement la distance de transformation. Si elles ne le sont pas, on dispose alors d'un algorithme heuristique, calculant une approximation de la distance en un temps raisonnable.

5.2.1 Préliminaires

L'ensemble des facteurs peut être partitionné en sous-ensembles de facteurs qui se chevauchent. Chaque sous-ensemble va contenir un ensemble de facteurs tel que pour tout couple de facteurs (f,h) de cet ensemble, $f <_s h$, on pourra trouver une suite de facteurs $(g_1, \ldots, g_n)$ de cet ensemble, $g_i <_s g_{i+1}$ pour tout $i \in 1..n-1$, tels que :

- $f \not<_o g_1$,
- $g_i \not<_o g_{i+1}$ pour tout $i \in 1..n-1$ et
- $g_n \not<_o h$.

Définition 5.9 *Un* ***sous-ensemble chevauchant*** *est une suite de facteurs* $(f_1, f_2, \ldots, f_n)$ *telle que* $f_1 <_s f_2 <_s \ldots <_s f_n$ *et telle que pour tout* $i \in 1..n$, *pour tout* $j \in i+1..n$, *il existe* k *indices* $m_1, \ldots, m_k$ *tels que* $f_i \not<_o f_{m_1} \not<_o f_{m_2} \cdots \not<_o f_{m_k} \not<_o f_j$.

L'ensemble des facteurs est donc l'union des ensemble chevauchants.

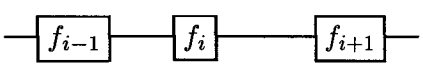
Exemple 5.3 *Avec les facteurs de l'exemple 5.1 l'ensemble des facteurs partitionné en sous-ensembles chevauchants est*

$$\{\{f_1,f_2,f_3\},\{f_4\},\{f_5\},\{f_6\},\{f_7,f_8,f_9,f_{10},f_{11},f_{12},f_{13},f_{14},f_{15},f_{16},f_{17},f_{18}\},\{f_{19}\}\}$$

Remarque 5.1 *Il peut exister un arc entre deux facteurs d'un même sous-ensemble chevauchant (c'est-à-dire que la relation $<_o$ est vérifiée). La propriété d'un sous-ensemble chevauchant est que deux facteurs successifs pris dans l'ordre $<_s$ ne vérifient pas la relation $<_o$. Cela n'implique pas que le premier facteur et le dernier du sous-ensemble ne vérifient pas la relation $<_o$.*

Pour un facteur quelconque de l'ensemble des facteurs, il est impossible de déterminer *a priori* si nous devons intégrer ce facteur dans la combinaison optimale. C'est pourquoi il existe un arc entre tous les facteurs tels que la relation $<_o$ soit vérifiée dans le graphe de scripts. Nous ne disposons pas d'une propriété locale nous permettant d'affirmer que la copie d'un facteur est plus intéressante que l'insertion du segment couvert par ce même facteur. Même si nous avions la propriété que $c(f_i) < i(f_i)$ pour tous les facteurs, cela ne suffirait pas à affirmer que la prise en compte de f_i dans un script aboutit à un script plus court que celui ne contenant pas f_i. Cette difficulté provient du fait que si nous choisissons d'insérer le segment couvrant f_i dans la séquence source plutôt que de le copier, nous ne remplaçons pas la copie de f_i par son insertion. Nous remplaçons l'insertion de f_{i-1} à f_i, la copie de f_i, l'insertion de f_i à f_{i+1} par une insertion allant de f_{i-1} à f_{i+1}.

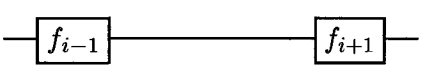
Exemple 5.4 *Illustrons cette idée par un schéma, imaginons donc avoir la configuration suivante:*



le script correspondant sera:

$$\ldots, c(f_{i-1}), i(f_{i-1}, f_i), c(f_i), i(f_i, f_{i+1}), c(f_{i+1}), \ldots$$

Si nous remplaçons la copie de f_i par l'insertion d'un segment équivalent, nous aurons alors:



et le script correspondant sera:

$$\ldots, c(f_{i-1}), i(f_{i-1}, f_{i+1}), c(f_{i+1}), \ldots$$

Examinons cela de plus près. Supposons que pour un facteur f_i nous ayons l'inégalité $c(f_i) < i(f_i)$. Cela implique

$$i(f_{i-1}, f_i) + c(f_i) + i(f_i, f_{i+1}) < i(f_{i-1}, f_i) + i(f_i) + i(f_i, f_{i+1})$$

Or, pour qu'il soit plus avantageux de copier f_i plutôt que de l'insérer, il faut que l'inégalité suivante soit vérifiée:

$$i(f_{i-1}, f_i) + c(f_i) + i(f_i, f_{i+1}) < i(f_{i-1}, f_{i+1})$$

Par conséquent, si la fonction d'insertion vérifie:

$$i(f_{i-1}, f_i) + i(f_i) + i(f_i, f_{i+1}) < i(f_{i-1}, f_{i+1})$$

alors il est effectivement moins coûteux de copier f_i que d'insérer le segment de T correspondant.

Nous voyons bien au travers de ces inégalités que la simple propriété que $c(f_i) < i(f_i)$ ne suffit pas à affirmer que la copie de f_i est plus intéressante. Si en plus de cette condition la fonction de coût d'une insertion est telle que $i(a,b) + i(b,c) < i(a,c)$ alors on peut choisir si f_i doit être

copié ou inséré. Il suffit donc que i ne vérifie pas l'inégalité triangulaire. Dans les autres cas, nous n'avons pas de caractérisation des fonctions i et nous ne savons pas conclure quant à l'opération qu'il faut appliquer à f_i.

Remarque 5.2 *Nous définirons très certainement le coût de l'opération d'insertion i pour qu'elle vérifie l'inégalité triangulaire. En effet, le cas contraire signifie que le coût associé à une insertion augmente proportionnellement de plus en plus avec la longueur de l'insertion, ce qui ne semble pas une hypothèse valide. En fait toute fonction convexe vérifiera l'inégalité triangulaire (en particulier les fonctions logarithmiques) et ne permettra pas de conclure.*

Nous avons vu qu'une borne maximale pour la complexité du graphe de script était $O(l^6)$ (où l était la longueur de la plus grande des deux séquences). Le calcul permettant d'obtenir cette borne est basé sur le calcul du nombre de facteurs qu'il peut exister au maximum entre deux séquences. L'algorithme que nous avons proposé précédemment, le graphe de scripts, nécessite que tous les facteurs soient présents dans le graphe. L'ensemble de tous les facteurs représente non seulement l'ensemble des facteurs communs mais également tous leurs sous-facteurs.

Exemple 5.5 *Considérons que nous recherchons l'ensemble de tous les facteurs de taille supérieure ou égale à 3. Si* $\langle ATTCCG, ATTCCG \rangle$ *est un facteur, il faut considérer également :*

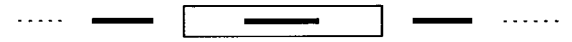
$\langle ATTCC, ATTCC \rangle$ $\langle ATTC, ATTC \rangle$, $\langle ATT, ATT \rangle$
$\langle TTCCG, TTCCG \rangle$, $\langle TTCC, TTCC \rangle$, $\langle TTC, TTC \rangle$
$\langle TCCG, TCCG \rangle$, $\langle TCC, TCC \rangle$
$\langle CCG, CCG \rangle$

Cela présente deux inconvénients. D'une part il est nécessaire de calculer l'ensemble de tous les facteurs communs entre les deux séquences. D'autre part, le graphe de scripts contient sans aucun doute un grand nombre de noeuds ne pouvant conduire à une solution optimale. Par conséquent il existe un grand nombre d'arcs inutiles qui sont relâchés.

Nous présentons dans cette section quelques conditions, qui, si elles sont vérifiées, conduisent à la construction et à la résolution d'un **graphe de scripts compact** permettant de calculer la solution optimale, où un nombre restreint de sous-facteurs sont présents. Dans le cas où ces conditions ne sont pas vérifiées, le graphe de scripts compact ne permet pas de calculer exactement la meilleure solution et devient un algorithme heuristique.

La définition du graphe de scripts compact est basée sur trois propriétés que nous supposerons vraies et dont nous allons détailler les implications dans la suite.

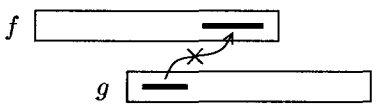
1. Si f est un facteur et g un sous-facteur de f, alors tout script incluant g et pour lequel g peut être remplacé par f sera d'un coût supérieur au même script où g est remplacé par f.



Cela signifie que l'on va toujours chercher à prendre les facteurs les plus grands dès que cela est possible. Cela nous permettra de ne pas inclure l'ensemble des sous-facteurs d'un facteur maximal[4] dans le graphe de scripts compact mais seulement une partie.

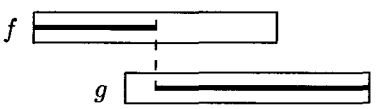
4. Un facteur maximal est tel qu'il n'est sous-facteur d'aucun autre facteur. Une définition formelle est donnée plus loin.

2. Si deux facteurs maximaux f et g se chevauchent ($f \not<_o g$) alors les arcs (g',f') reliant un sous-facteur g' de g et un sous-facteur f' de f ne conduisent pas à une solution optimale.



On préférera remplacer les opérations de copie de g', d'insertion entre g' et f' et de copie de f' par une seule opération de copie d'un facteur de f couvrant la même portion de la séquence cible.

3. Si deux facteurs maximaux f et g se chevauchent ($f \not<_o g$) alors il existe un seul arc (f',g') reliant un sous-facteur f' de f et un sous-facteur g' de g qui puisse aboutir à une solution optimale.



Il suffira alors d'inclure dans le graphe de scripts les deux seuls sous-facteurs f' et g', les autres ne permettant pas d'accéder à un script de poids plus faible.

Nous commençons par donner quelques définitions qui vont nous aider à préciser ces trois conditions que nous étudirons ensuite. Nous terminons par donner un algorithme permettant le calcul de la distance de transformation avec le graphe de scripts compact sans construction effective du graphe.

5.2.2 Définitions

Nous introduisons d'abord formellement les notions de facteur maximal et de sous-facteur évoquées plus haut. Pour cela, nous définissons la notion d'inclusion d'un facteur dans un autre. Un facteur sera inclus dans un autre s'il couvre respectivement dans la source et dans la cible un sous-segment du premier et si de plus il existe le même décalage entre les segments dans les deux séquences.

Définition 5.10 *Soit $f = \langle f_s,f_t \rangle$ et $g = \langle g_s,g_t \rangle$ deux facteurs. On dira que g est* ***inclus*** *dans f, et on notera $g \subset f$, si et seulement si $f_s = ug_su'$, $f_t = vg_tv'$ et $|u| = |v|$, $|u'| = |v'|$.*

Définition 5.11 *Un* ***sous-facteur*** *f d'un facteur Φ est tel que $f \subset \Phi$.*

Définition 5.12 *Un facteur f est un* ***facteur maximal*** *si il n'existe pas de facteur f' tel que f soit inclus dans f'.*

Pour un ensemble de facteurs F, on notera $F_{\max}$l'ensemble des facteurs maximaux associé. On notera MAX(f) le facteur maximal associé au facteur f. Une illustration de ces définitions est donnée figure 5.3. Afin de distinguer un facteur maximal d'un sous-facteur dans les schémas, les facteurs maximaux seront représentés par un cadre.

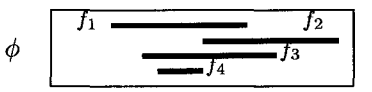


FIG. 5.3 – *Illustration des sous-facteurs et des facteurs maximaux. f_1,f_2,f_3 et f_4 sont des sous-facteurs de ϕ. f_4 est inclus dans f_3. f_3 n'est pas maximal car il est inclus dans ϕ qui lui est maximal.*

Exemple 5.6 *Si l'on considère l'ensemble des facteurs de l'exemple 5.1, l'ensemble des facteurs maximaux est :*

1 = 27 7 6
2 =14 15 5
3 =43 26 5
4 =43 35 5
5 =36 42 6
6 =44 43 5
7 =38 46 7
8 =45 46 5
9 =23 48 5
10=14 54 5

Nous donnons maintenant les définitions de facteur préfixe et de facteur suffixe. C'est une extension de la notion usuelle utilisée sur les mots. La notion de suffixe et de préfixe est ici appliquée au segment du facteur relatif à la séquence cible.

Définition 5.13 *Un facteur f est **préfixe** d'un facteur g si f^t est préfixe de g^t.*

Définition 5.14 *Un facteur f est **suffixe** d'un facteur g si f^t est suffixe de g^t.*

Nous introduisons maintenant deux caractéristiques des arcs du graphe de scripts : les **arcs ascendants** et les **arcs descendants**. La dénomination que nous avons choisi provient de la représentation que nous avons adoptée lorsqu'il s'agit de représenter les facteurs dans la séquence cible. Nous représentons systématiquement ceux-ci par un segment délimitant la portion de la séquence cible qui est couverte et, chaque fois qu'un facteur en chevauche un autre, par exemple $f <_s g$ et $f \not<_o g$, nous représentons le facteur g en dessous du facteur f. Un exemple a été donné figure 5.1.

Un arc ascendant est un arc dont le facteur maximal du facteur cible[5] de l'arc précède et chevauche le facteur maximal du facteur source[6] de l'arc. Un arc descendant est un arc dont le facteur maximal du facteur source de l'arc précède et chevauche le facteur maximal du facteur cible de l'arc (voir figure 5.4).

Définition 5.15 *Un **arc ascendant** est un arc (f,g) tel que* $\text{MAX}(g) <_s \text{MAX}(f)$ *et* $\text{MAX}(g) \not<_o \text{MAX}(f)$.

5. Le facteur cible d'un arc est le facteur représenté par le sommet but de l'arc
6. Le facteur source d'un arc est le facteur représenté par le sommet origine de l'arc

Définition 5.16 *Un arc* ***descendant*** *est un arc* (f,g) *tel que* $\text{MAX}(f) <_s \text{MAX}(g)$ *et* $\text{MAX}(f) \not<_o \text{MAX}(g)$.

Remarque 5.3 *On peut noter que les définitions d'arc ascendant et descendant ne permettent pas de partitionner l'ensemble des arcs en deux catégories. Il existe des arcs qui ne sont ni ascendant, ni descendant. En effet, ces arcs se limitent aux facteurs tels que leurs facteurs maximaux respectifs se chevauchent. On ne trouvera donc des arcs ascendants et descendants qu'entre des facteurs appartenant à un même sous-ensemble chevauchant.*

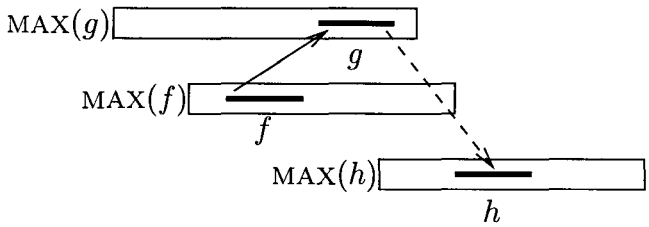


FIG. 5.4 – *Exemple d'arc ascendant (en trait plein) et descendant (en trait discontinu). Les facteurs maximaux sont représentés par les boîtes. Les sous-facteurs par les traits. Les facteurs maximaux respectent la représentation choisie :* $\text{MAX}(g)$ *est représenté au dessus de* $\text{MAX}(f)$ *puisque* $\text{MAX}(g) <_s \text{MAX}(f)$.

Ayant donné ces définitions, nous poursuivons maintenant par des propriétés permettant d'éliminer par avance des scripts conduisant à une solution non optimale. Les propriétés portent sur les arcs du graphe de scripts mais nous avons clairement établi qu'un script, un $f-$script ou un chemin de a (le noeud source) à z (le noeud puits) dans le graphe de scripts sont trois représentations du même objet.

5.2.3 Propriétés des scripts non optimaux

Optimalité des facteurs maximaux

Ce premier critère, visant à limiter l'espace de recherche pour le script minimal, est basé sur l'idée naturelle qui consiste à penser que la présence d'un long facteur dans un script aboutit à un script de poids plus faible que le même script où on aurait remplacé ce long facteur par un de ses sous-facteurs. Formalisée dans le vocabulaire que nous avons introduit, cette idée s'énonce : **un facteur maximal est meilleur que ses sous-facteurs**. Pour que cela soit vérifié, il est nécessaire qu'une condition sur les fonctions de poids de copie c et d'insertion i soit vérifiée.

Condition 5.1 (des facteurs maximaux) *Soient trois facteurs* f,g,h *tels que* $f <_o g <_o h$, *on dit que* f,g,h *vérifient la* ***condition des facteurs maximaux*** *si pour tout sous-facteur* $g' \subset g$:

$$i(f,g) + c(g) + i(g,h) < i(f,g') + c(g') + i(g',h)$$

Nous montrons maintenant qu'un script contenant un sous-facteur d'un facteur maximal est d'un poids supérieur à celui contenant le facteur maximal associé (sous la réserve que la condition des facteurs maximaux soit vérifiée pour tout triplet de facteurs).

Lemme 5.3 (des facteurs maximaux) *Soit $\phi_1 = f_1,\dots,f_{i-1},f_i,f_{i+1},\dots,f_n$ un chemin. Si la condition des facteurs maximaux est vérifiée entre tout triplet de facteurs, alors, s'il existe un facteur f'_i tel que $f_i \subset f'_i$ et tel que $f_{i-1} <_o f'_i <_o f_{i+1}$, le chemin $\phi_2 = f_1,\dots,f_{i-1},f'_i,f_{i+1},\dots,f_i$ sera meilleur que ϕ_1.*

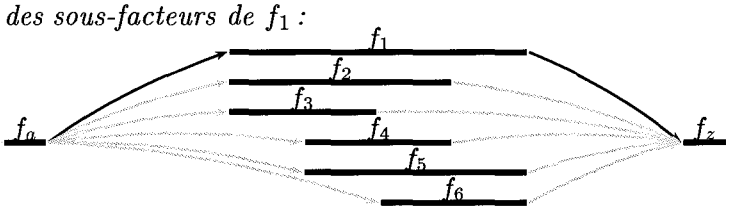
Preuve 5.3

$w(\phi_2) \leq w(\phi_1)$ si l'inégalité suivante est vérifiée :

$$i(f_{i-1},f'_i) + c(f'_i) + i(f'_i,f_{i+1}) \leq i(f_{i-1},f_i) + c(f_i) + i(f_i,f_{i+1})$$

Comme la condition 5.1 est vérifiée pour f_{i-1}, f'_i, f_{i+1} et f_i, l'inégalité est vraie. □

Exemple 5.7 *Si on dispose du graphe de scripts suivant où f_1 est un facteur maximal et $f_2,\dots,f_6$ sont des sous-facteurs de f_1 :*



seuls les arcs en trait foncé permettront d'aboutir à un chemin optimal.

Inutilité des arcs ascendants

Nous allons exprimer ici le fait que si deux facteurs maximaux se chevauchent tels que le premier précède le second, alors un chemin optimal ne peut contenir d'arc joignant un sous-facteur du second et un sous-facteur du premier.

De manière informelle, cela signifie que nous allons préférer prendre directement un plus grand sous-facteur du premier facteur maximal plutôt que de passer par un sous-facteur du second. La condition suivante exprime qu'il vaut mieux effectuer une copie d'un seul facteur que deux copies et éventuellement une insertion pour couvrir une même une partie de la séquence cible.

Condition 5.2 (des arcs ascendants) *Soient un couple de facteurs f,g tels que $f <_o g$ et $\text{MAX}(g) <_s \text{MAX}(f)$ et h un facteur débutant aux mêmes positions que f et de longueur la différence entre les positions de fin de g et de début de f ($h = \langle f^s,f^t,(g^t + g^l) - f^t\rangle$), on dit que la **condition des arcs ascendants** est vérifiée si :*

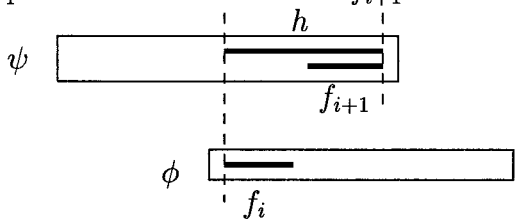
$$c(h) < c(f) + i(f,g) + c(g)$$

Lemme 5.4 (des arcs ascendants) *Soit $a,f_1,\dots,f_i,f_{i+1},\dots,f_n,z$ un chemin dans un graphe de scripts $G = (V,E,w)$ tel que $(f_i,f_{i+1}) \in V$ est un arc ascendant, alors ce chemin n'est pas optimal si la condition des arcs ascendants est vérifiée.*

Preuve 5.4

On sait que (f_i,f_{i+1}) est un arc ascendant. Notons $\psi = \text{MAX}(f_{i+1})$ et $\phi = \text{MAX}(f_i)$.
Soit $h \subset \psi$ un facteur débutant à la même position que f_i et se terminant à la même

position que f_{i+1} dans la séquence source. Il est clair que le facteur h existe[7]. En effet, puisque $\psi \not<_o \phi$ il existe une partie commune à ces deux facteurs et ψ débute avant ϕ. Comme de plus (f_i,f_{i+1}) est un arc ascendant, f_i débute et se termine avant f_{i+1}. Il existe donc nécessairement un ou plusieurs sous-facteurs de ψ débutants à la même position que f_i. Parmi ces sous-facteurs, il en existe un se terminant à la même position que f_{i+1} puisque ceux-ci débutent avant f_{i+1}. On choisit celui-là pour h.



Montrer que l'arc (f_i,f_{i+1}) ne peut conduire à une solution optimale revient à montrer qu'il existe un chemin meilleur. Le chemin passant par h sera de poids inférieur au chemin passant par f_i puis f_{i+1} si l'inégalité suivante est vérifiée:

$$w(h,f_{i+2}) < w(f_i,f_{i+1}) + w(f_{i+1},f_{i+2})$$

ce qui est équivalent à montrer que:

$$c(h) + i(h,f_{i+2}) < c(f_i) + i(f_i,f_{i+1}) + c(f_{i+1}) + i(f_{i+1},f_{i+2})$$

Comme f_{i+1} et h terminent à la même position, $i(f_{i+1},f_{i+2}) = i(h,f_{i+2})$, il faut donc que:

$$c(h) < c(f_i) + i(f_i,f_{i+1}) + c(f_{i+1})$$

ce qui est donné par la condition 5.2. □

Unicité des arcs descendants

Lorsque deux facteurs maximaux se chevauchent, il existe des arcs descendants entre ces facteurs. Le lemme que nous énonçons nous permet d'affirmer que les seuls arcs descendants intéressants sont ceux pour lesquels le script associé ne contiendra pas d'opération d'insertion entre les deux opérations de copie des facteurs source et but de l'arc.

Condition 5.3 (unicité des arcs descendants) *Soient un couple de facteurs f',g' tels que $f' <_o g'$ et $\text{MAX}(f') \not<_o \text{MAX}(g')$, avec $f \subset f'$ débutant à la même position que f', se terminant avant f' et $g \subset g'$ se terminant à la même position que g', débutant après le début g', on dit que la **condition d'unicité des arcs descendants** est vérifiée si:*

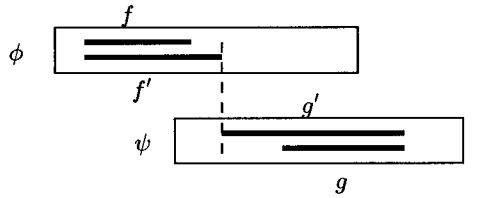
$$c(f) + i(f,g) + c(g) \geq c(f') + c(g')$$

Lemme 5.5 (unicité des arcs descendants) *Soient f et g deux facteurs tels que (f,g) soit un arc descendant et que $f <_o g$. Soient ϕ et ψ les facteurs maximaux respectifs de f et g. Si la condition de l'unicité des arcs descendants est vérifiée alors un chemin incluant f et g n'est optimal que si f se termine à la position où commence g.*

7. L'existence de h peut dépendre de la manière dont ont été définis les facteurs. Pour le cas particulier de facteurs exacts, on est certain de cette existence.

Preuve 5.5

Supposons qu'il existe une zone d'insertion entre f et g. Soient f' et g' deux sous-facteurs de ϕ et ψ tels que f' débute à la même position que f et se termine là ou débute g', g' se terminant à la même position que g. Il est clair que f' et g' existent toujours puisque ϕ et ψ se chevauchent[8].



Si l'inégalité suivante est vérifiée :

$$c(f) + i(f,g) + c(g) \geq c(f') + c(g')$$

alors il est plus intéressant de choisir le couple f' et g' plutôt que f et g. La condition 5.3 nous permet de vérifier cette inégalité. □

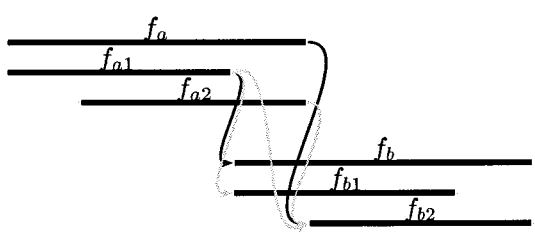
On peut déduire de cela les meilleurs sous-facteurs de deux facteurs qui se chevauchent. Le lemme des facteurs maximaux indique que les sous-facteurs doivent être choisis les plus grands possibles, et donc qu'il ne doit pas exister d'insertion entre eux.

Lemme 5.6 (des facteurs réduits) *Soient f_1,f_2,f_3,f_4 quatre facteurs tels que $f_1 <_o f_2 \not<_o f_3 <_o f_4$. Soient $f'_2 \subset f_2$ et $f'_3 \subset f_3$. f_1,f'_2,f'_3,f_4 ne peut être optimal si f'_2 est un préfixe de f_2, f'_3 un suffixe de f_3 et que f'_2 se termine là où débute f'_3.*

Preuve 5.6

Supposons que f'_2 ne soit pas un préfixe de f_2. Alors d'après le lemme 5.3 il existe un sous-facteur de f_2, débutant avant f'_2 et tel que le chemin passant par ce sous-facteur est meilleur. Par itération, le sous-facteur de f_2 maximisant le chemin est un préfixe de f_2. Pour un facteur f'_3 donné, les mêmes arguments montrent que f'_2 doit se terminer là où débute f'_3. La démonstration est similaire pour f'_3. □

Exemple 5.8 *Si nous disposons de l'ensemble de facteurs suivant :*



où f_a et f_b sont des facteurs maximaux, f_{a1} et f_{a2} des sous-facteurs de f_a et f_{b1} et f_{b2} des sous-facteurs de f_b. Seuls les arcs noirs, qui relient un préfixe de f_a et un suffixe de f_b ne nécessitant pas d'insertion entre-eux, devront être intégrés dans le graphe, les autres ne pouvant amener à une solution optimale.

8. Même remarque que précédemment, cela dépend de la définition des facteurs mais cela est vrai pour des facteurs exacts.

Facteurs réduits

Nous avons vu d'après le lemme 5.6 que le meilleur choix pour optimiser un chemin f_1,f_2,f_3,f_4 tels que $f_1 <_o f_2 \not<_o f_3 <_o f_4$ passant par des sous-facteurs de f_2 et f_3 est de choisir un préfixe de f_2 et un suffixe de f_3 ne nécessitant pas d'insertion entre eux.

Parmi ces couples de préfixes et de suffixes, il en existe au moins un meilleur que les autres. Nous appelons ce meilleur choix de sous-facteurs le **facteur réduit** de f_2 et f_3 et nous le notons RÉDUIT(f_2,f_3). Pour trouver RÉDUIT(f_2,f_3), il est nécessaire de minimiser la somme

$$c(f_2') + c(f_3')$$

avec f_2' un préfixe de f_2 et f_3' un suffixe de f_3. RÉDUIT(f_2,f_3) est une suite de deux facteurs non chevauchants.

Il est possible de généraliser à plusieurs facteurs chevauchants. Si on dispose maintenant d'un facteur réduit r = RÉDUIT(f,g) = $\{f_1,g_1\}$, et que h est un facteur tel que $g \not<_o h$ et $g <_s h$ alors on doit calculer

$$\text{RÉDUIT}(r,h) = \text{RÉDUIT}(\{f_1,g_1\},h) = \text{RÉDUIT}(\text{MAX}(f_1),\text{MAX}(g_1),h) = \text{RÉDUIT}(f,g,h)$$

qui doit correspondre à la meilleure sélection de sous-facteurs de f,g et h. Il faut cette fois minimiser la somme :

$$c(f') + c(g') + c(h')$$

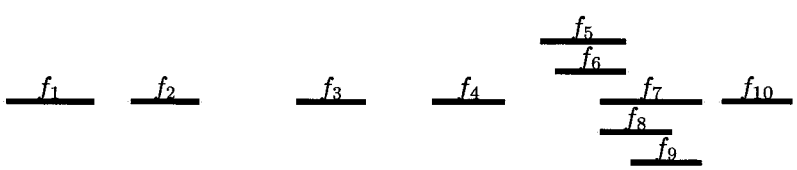
avec f' un préfixe de f, h' un suffixe de h et g' un sous-facteur de g se terminant là où débute h' et débutant là où se termine f'.

De manière générale, pour calculer RÉDUIT(f_1,f_2,...,f_n), nous devons minimiser

$$\sum_{i=1}^{n} c(f_i')$$

avec f_1' un préfixe de f_1, f_n' un suffixe de f_n et f_j, $j \in 1..n-1$ se termine là où débute f_{j+1}.

Exemple 5.9 *Reprenons l'ensemble des facteurs maximaux de l'exemple 5.6. La représentation de ces facteurs est la suivante :*



On a $f_5 \not<_o f_6$ mais il n'est pas possible de trouver un préfixe de f_5 et un suffixe de f_6 satisfaisant aux contraintes de longueur que nous avons fixées (dans notre exemple, la longueur minimale des facteurs doit être de 5). Par conséquent RÉDUIT(f_5,f_6) = $\emptyset$. *De même on a* RÉDUIT(f_5,f_8) = RÉDUIT(f_6,f_8) = $\emptyset$. *Par contre, il existe une réduction possible pour (f_5,f_7) et (f_6,f_7). On a :* RÉDUIT(f_5,f_7) = {(36,42,5),(38,46,7)} *(l'autre choix* {(36,42,6),(39,47,6)} *est plus coûteux) et* RÉDUIT(f_6,f_7) = {(44,43,5),(40,48,5)} *(seul choix possible).*

5.2.4 Définition du graphe de scripts compact

Nous pouvons maintenant décrire une méthode qui va permettre de construire un graphe de scripts basé sur les facteurs maximaux. Dans ce graphe, l'ensemble des noeuds est limité aux facteurs maximaux et aux facteurs réduits entre les facteurs maximaux se chevauchant. On suppose que les conditions des facteurs maximaux (5.1), des arcs ascendants (5.2) et d'unicité des arcs ascendants (5.3) sont vérifiées.

Supposons que nous disposions de l'ensemble des facteurs maximaux ordonné selon la relation $<_s$. Nous cherchons à calculer l'ensemble des arcs reliant ces facteurs.

Soit f le facteur pour lequel nous calculons l'ensemble de ses successeurs. Il n'est pas nécessaire de s'occuper des facteurs précédant f puisque le lemme des arcs ascendants (lemme 5.4) montre qu'il n'existe pas d'arc entre deux sous-facteurs tels que leur facteurs maximaux respectifs ne sont pas dans le même ordre pour la relation $<_s$. Nous ne nous intéressons donc qu'aux facteurs précédés par f.

Soit g un facteur tel que $f <_s g$.

- Si $g = \text{RÉDUIT}(h_1,h_2,\ldots,h_n)$, ses prédécesseurs sont les mêmes que ceux de h_1 et sont mis à jour lors de la création de g, il n'y a pas de traitement à effectuer,
- Si g n'est pas un facteur réduit,
  - Supposons que $f <_o g$ alors il doit exister un arc entre f et g. Grâce au lemme des facteurs maximaux (lemme 5.3) nous pouvons affirmer que les arcs entre tout sous-facteur de f et tout sous-facteur de g sont inutiles.
  - Supposons maintenant que $f \not<_o g$. Pour aboutir à la solution optimale il va être nécessaire de créer des arcs descendants entre f et g.
    - Si $f = \text{RÉDUIT}(f_1,f_2,\ldots,f_n)$ alors il est nécessaire de calculer $r = \text{RÉDUIT}(f_1,f_2,\ldots,f_n,g)$. Il faut ensuite insérer ce nouveau facteur dans la liste des facteurs. Il suffit de regarder les prédécesseurs de f_1 pour obtenir les prédécesseurs de r. Ainsi le poids de r sera celui de f_1 et r est "virtuellement" déjà inspecté pour tous les facteurs qui le précède.
    - Si f n'est pas un facteur réduit, alors il faut créer $\text{RÉDUIT}(f,g)$ et l'insérer dans la liste des facteurs.

La définition du graphe de scripts compact est donnée ci-dessous. Un exemple basé sur l'ensemble des facteurs de l'exemple 5.6 est donné figure 5.5. L'algorithme 5.4 présente un algorithme correspondant à la description informelle donnée ci-dessus.

Définition 5.17 *Un* ***graphe de scripts compact*** G_c *est un triplet* (V,E,w) *où*

- $R = \{\text{RÉDUIT}(f_1,f_2,\ldots,f_n) \mid f_1 \not<_o f_2 \not<_o \cdots \not<_o f_n\}$,
- $V = F_{max} \cup \{a,z\} \cup R$ *est l'ensemble des sommets,*
- $E = \{(f,g) \mid f,g \in V, f <_o g\}$ *est l'ensemble des arcs,*
- w *est une fonction qui associe un poids à chaque arc : pour tout arc* (f,g) *de* E, $w((f,g)) = c(f) + i(f,g)$.

Remarque 5.4 $r = \text{RÉDUIT}(f_1,f_2,\ldots,f_n)$ *correspond à une optimisation pour un chemin particulier. Il n'y a pas de raison pour qu'un chemin optimal, autre que celui pour lequel* r *a été*

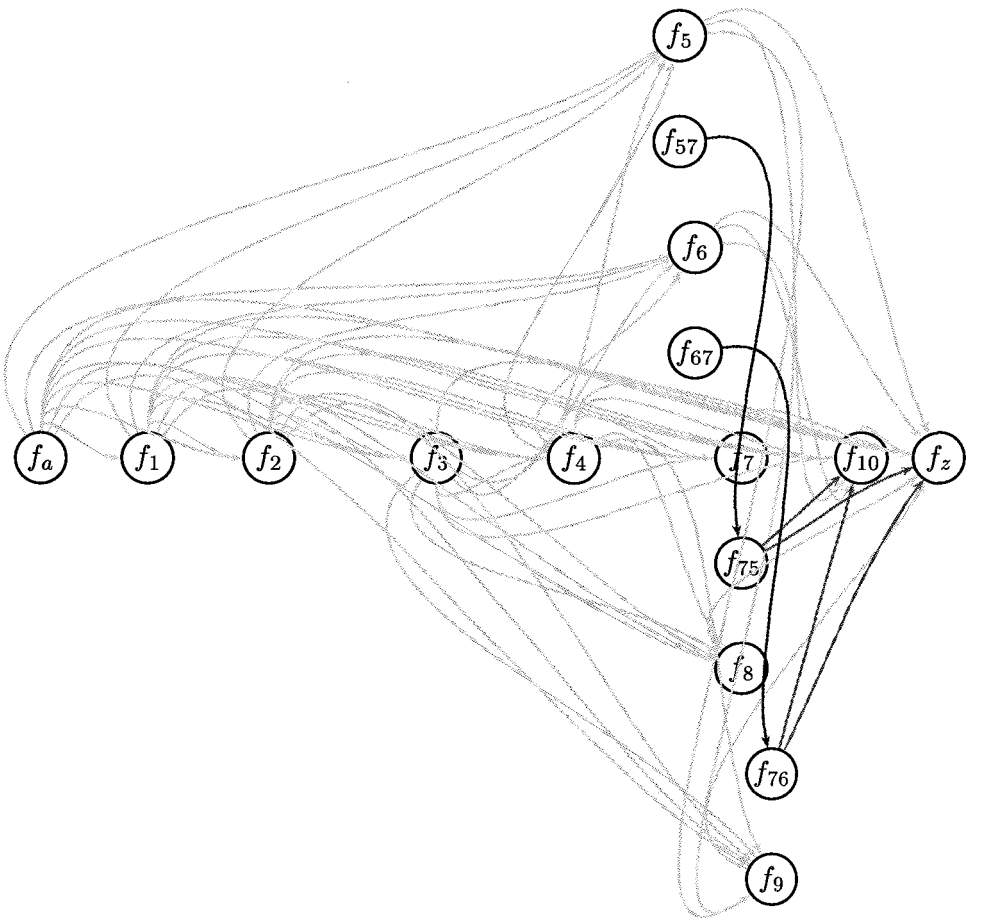


FIG. 5.5 – *Le graphe de scripts compact pour l'ensemble des facteurs de l'exemple 5.6 (qui est l'ensemble des facteurs maximaux de l'exemple 5.1). Pour cet ensemble seules les réductions suivantes donnent lieu à de nouveaux facteurs (voir exemple 5.9) :* RÉDUIT(f_5,f_7) = {(36,42,5),(38,46,7)} = {f_{57},f_{75}} *et* RÉDUIT(f_6,f_7) = {(44,43,5),(40,48,5)} = {f_{67},f_{76}}. *Ces quatres nouveaux facteurs apparaissent dans le graphe. Il n'y a pas d'arc aboutissant à f_{57} et à f_{67} puisqu'ils ont respectivement les mêmes prédecesseurs que f_5 et f_6 (qui sont déjà calculés lorsque f_{57} et f_{67} sont créés). Il n'existe qu'un seul arc reliant f_{57} et f_{75}, de même pour f_{67} et f_{76}. f_{75} et f_{76} sont reliés de manière classique (en respectant la relation $<_o$) à f_{10} et z. Le graphe comporte 84 arcs (contre 157 pour le graphe de scripts).*

ALG 5.4 MEILLEURSCRIPTCOMPACT : Algorithme de calcul de la distance de transformation utilisant les lemmes 5.3, 5.4, 5.5, 5.6.

Entrée: $F_{\max}$un ensemble de facteurs maximaux ordonné selon la relation $<_s$
Sortie: la distance de transformation (par copie de facteurs)

$V \leftarrow F_{\max} \cup \{a,z\}$
{*initialisation du coût des sommets*}
pour tout $f \in V$ **faire**
  $w(f) \leftarrow +\infty$
fin pour
$w(a) \leftarrow 0$
{*calcul du meilleur chemin*}
pour tout $f \in V$ **faire**
  pour tout $g \in V$ tel que $f <_s g$ **faire**
    si g n'est pas un facteur réduit **alors**
      si $f <_o g$ **alors**
        RELÂCHER(f,g)
      sinon
        {*nous devons créer la réduction de* f *par* g}
        $r \leftarrow$ RÉDUIT(f,g)
        $w(r) \leftarrow w(f)$
        ajouter r à V en respectant la relation $<_s$
      fin si
    fin si
  fin pour
fin pour
retourner $w(z)$

calculé emprunte l'un des facteurs de r. Si c'est le cas, c'est que ce facteur a été calculé pour une autre réduction. Par conséquent, (f_i,f_{i+1}), $i \in 1..n-1$, *est l'unique arc sortant de* f_i *et aboutissant à* f_{i+1}. *Dans la suite, on assimilera la suite de facteurs* RÉDUIT$(f_1,f_2,\ldots,f_n)$ *à un seul facteur.*

L'utilisation de cette méthode nécessite de calculer des facteurs réduits. Rappelons que la recherche de RÉDUIT$(f_1,f_2,\ldots,f_n)$ nécessite de minimiser $\sum_{i=1}^{n} c(f_i')$ (où f_i' est un sous-facteur de f_i, voir page 107). Ce calcul peut être coûteux, en particulier si le calcul de RÉDUIT$(f_1,f_2,\ldots,f_{n+1})$ ne tient pas compte du calcul de RÉDUIT$(f_1,f_2,\ldots,f_n)$.

Si la fonction de réduction possède la propriété suivante : pour tout triplet de facteurs f,g,h, $f <_s g <_s h$, $f \not<_o g$ et $g \not<_o h$,

$$\text{RÉDUIT}(f,g,h) = \text{RÉDUIT}(g',h)$$

avec

$$\text{RÉDUIT}(f,g) = \{f',g'\}$$

alors on observera un gain de temps pour le calcul de la distance de transformation. En effet, d'une part il ne sera pas nécessaire de se souvenir de la liste des facteurs maximaux ayant permis d'engendrer un facteur réduit, d'autre part le calcul de la réduction ne dépendra que de deux facteurs.

5.3 Cas d'un ensemble de facteurs valides

Lorsque nous recherchons un script, nous nous basons sur un ensemble de facteurs (ici il n'est pas question de facteurs maximaux). Nous effectuons ensuite un choix d'une combinaison de facteurs minimisant la fonction de poids que nous avons définie précédemment. Nous avons vu qu'il était difficile de déterminer par avance si l'intégration d'un facteur à une combinaison diminuait ou augmentait le coût de cette combinaison.

Nous étudions ici le cas particulier où l'on a cette propriété. Nous supposons donc disposer d'un ensemble de facteurs où il ne reste que des facteurs dont on sait par avance que leur intégration à une combinaison améliore son coût.

Il est très rare de disposer d'un tel ensemble de facteurs. On peut imaginer qu'un ensemble de facteurs est proposé par le biologiste qui nous a indiqué qu'un nombre maximum de ces facteurs devaient être inclus dans un script. En fait, l'algorithme développé dans cette section sera le plus souvent utilisé comme une heuristique. On supposera alors que tous les facteurs sont valides, même si la condition sur les fonctions de poids que doivent vérifier les facteurs n'est pas vraie. Nous verrons chapitre 6 que si la longueur des facteurs est suffisamment grande, le script valide optimal correspond au script optimal.

5.3.1 Condition des facteurs valides

La condition suivante exprime le fait qu'il est plus avantageux d'ajouter un facteur dans une combinaison de facteurs dès que cela est possible. C'est-à-dire dès que $f <_o h$ et qu'il existe un facteur g tel que $f <_o g <_o h$.

Condition 5.4 (des facteurs valides) *Soient trois facteurs f,g,h tels que $f <_o g <_o h$, on dit que la* ***condition des facteurs valides*** *est vérifiée si :*

$$i(f,g) + c(g) + i(g,h) < i(f,h)$$

Nous pouvons supposer disposer d'un ensemble F_v de **facteurs valides**. C'est-à-dire un ensemble de facteurs tels que, chaque fois qu'un facteur peut être intégré à un script, il doit l'être. Cela signifie qu'un script ne prenant pas en compte un facteur aura un poids plus grand que le script le prenant en compte. Et donc, la condition 5.4 doit être vérifiée pour tout triplet de facteurs (f,g,h) de l'ensemble tel que $f <_o g <_o h$.

Définition 5.18 *Un ensemble de* ***facteurs valides*** *F_v est un ensemble de facteurs tel que pour tout triplet de facteurs f,g,h de F_v tels que $f <_o g <_o h$, la condition 5.4 est vérifiée.*

Un script bâtit sur un ensemble de facteurs valides est appelé un **script valide**. Grâce à la propriété des facteurs valides, on peut construire un **graphe de scripts valides** qui ne contiendra pas tous les scripts possibles, certains chemins étant associés à des scripts non optimaux. Les scripts valides sont un sous-ensemble des scripts tel que le retrait d'un facteur (c'est-à-dire d'une opération de copie d'un facteur) d'un script implique que le poids de ce script augmente.

Lemme 5.7 (des facteurs valides) *Soit g un facteur. Si, pour tous facteurs f,h tels que $f <_o g <_o h$, la condition des facteurs valides est vérifiée, alors tout script contenant g est de poids inférieur au même script ne contenant pas g*[9].

La preuve est directe. Il suffit de comparer le poids d'un script contenant g et le poids d'un script ne le contenant pas. On aboutit immédiatement à l'inégalité de la condition 5.4.

Si nous considérons une combinaison de facteurs $(f_1,f_2,\ldots,f_n)$ d'un ensemble de facteurs valides tels que $f_i <_o f_{i+1}$ pour tout i appartenant à $1..n-1$, alors le script associé forme un sous-script d'un script valide. En effet, à partir de la combinaison $(f_1,f_2,\ldots,f_n)$ il suffit d'ajouter avant f_1, entre chaque f_i et f_{i+1} et après f_n les facteurs disponibles dans l'ensemble et ne les chevauchant pas. Ainsi, entre f_i et f_{i+1} on ajoutera la suite de facteurs $g_{i,1},g_{i,2},\ldots,g_{i,n_i}$ telle que $f_i <_o g_{i,1} <_o g_{i,2} <_o \cdots <_o g_{i,n_i} <_o f_{i+1}$ et qu'il n'existe pas de facteur h tel que $f_i <_o h <_o g_{i,1}$ ou $g_{i,j} <_o h <_o g_{i,j+1}$ ou $g_{i,n_i} <_o h <_o f_{i+1}$.

5.3.2 Graphe de scripts valides

Nous poursuivons en définissant la notion de graphe de scripts valides.

Définition 5.19 *Un* ***graphe de scripts valides*** *G_v est un triplet (V,E,w) où*

- *$V = F_v \cup \{a,z\}$ est l'ensemble des sommets,*
- *$E = \{(f,g) \mid f,g \in V, f <_o g, \nexists h \in V, f <_o h <_o g\}$ est l'ensemble des arcs,*
- *w est une fonction qui associe un poids à chaque arc : pour tout arc (f,g) de E, $w((f,g)) = c(f) + i(f,g)$.*

9. Un script ne contenant pas g est entendu comme le même script duquel on retire le facteur g.

Le graphe de scripts valides supprime les arcs qui ne permettent pas d'aboutir à une solution optimale : il n'existe pas d'arcs reliant deux facteurs de deux sous-ensembles chevauchant séparés par un autre sous-ensemble chevauchant (voir figure 5.6). De plus, si il existe deux facteurs f_i et f_{i+1} d'un sous-ensemble chevauchant tels que $f_i <_o f_{i+1}$, alors il n'existe pas d'arc reliant f_i à l'un des facteurs du sous-ensemble suivant et d'arc reliant un des facteurs du sous-ensemble précédent à f_{i+1} (voir figure 5.7). Cela est formalisé plus loin.

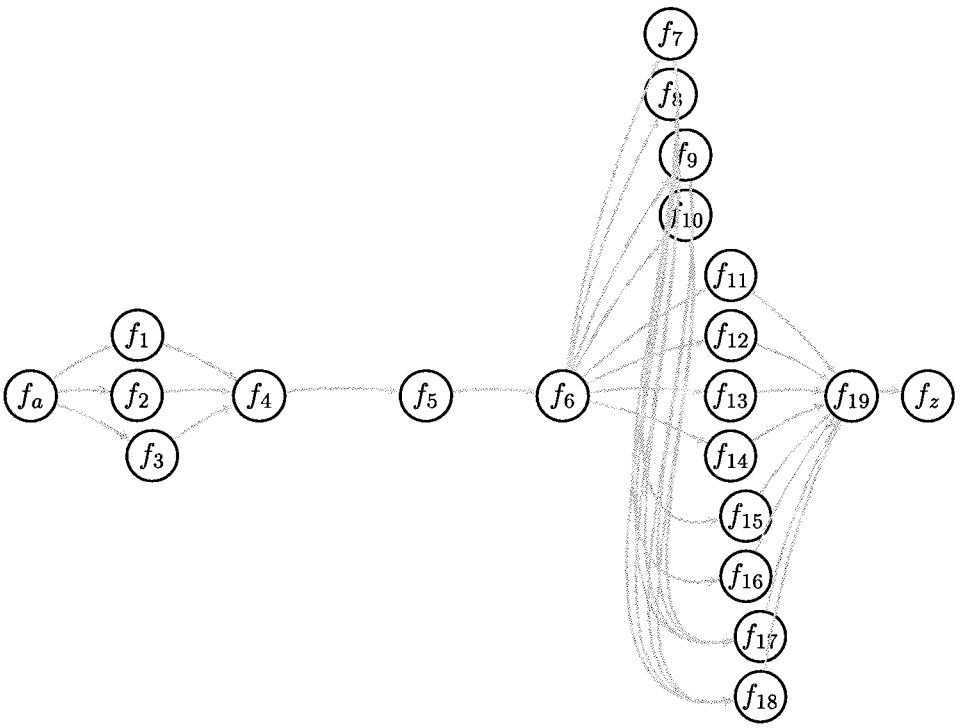


FIG. 5.6 – *Le graphe de scripts valides pour l'ensemble de facteurs de l'exemple 5.1. Le graphe de scripts valides ne contient pas d'arcs entre des ensembles chevauchants séparés par d'autres ensembles chevauchants (il n'existe pas d'arc reliant* $\{f_1,f_2,f_3\}$ *et* f_5*). Il existe des arcs internes entre* $\{f_7,\ldots,f_{10}\}$ *et* $\{f_{15},\ldots,f_{18}\}$*, par conséquent* f_6 *n'est relié qu'à* $\{f_7,\ldots,f_{14}\}$ *et les arcs aboutissants à* f_{19} *sont* $\{f_{11},\ldots,f_{18}\}$*. Le graphe comporte 41 arcs (contre 157 pour le graphe de scripts et 84 pour le graphe de scripts compact).*

Proposition 5.5 *Un chemin dans le graphe de scripts valides représente un script valide.*

Preuve 5.5

Considérons une suite de facteurs $a,f_1,f_2,\ldots,f_n,z$ dans un graphe de scripts valides. Notons a, f_0 et z, f_{n+1}. Par définition du graphe de scripts valides, pour chaque arc (f_i,f_{i+1}), i appartenant à $0..n$ il n'existe pas de facteur h tel que $f_i <_o h <_o f_{i+1}$. $(f_1,f_2,\ldots,f_n)$ est donc un script valide. □

Proposition 5.6 *Soit* F_v *un ensemble de facteurs valides. Un chemin de poids minimal de a à*

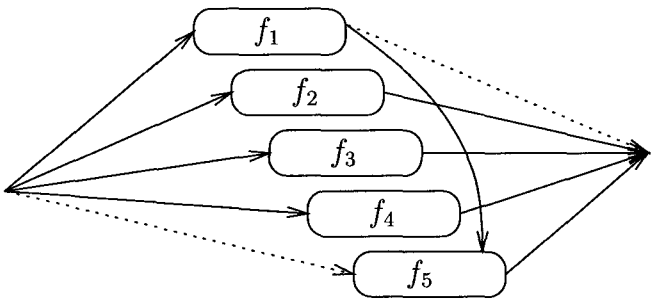


FIG. 5.7 – *Représentation des arcs entrant et sortant du sous-ensemble chevauchant* $f_1,f_2,\ldots,f_5$. *Les arcs en pointillés n'existent pas dans le graphe de scripts valides car tout script contenant* f_1 *et* f_5 *sera meilleur que celui ne contenant que* f_1 *ou* f_5. *L'existence d'un arc entre* f_1 *et* f_5 *implique la suppression des arcs vers* f_5 *et des arcs sortant de* f_1.

z dans le graphe de scripts valides construit sur F_v *est le même que le chemin de poids minimal de a à z dans le graphe de scripts construit sur* F_v.

Preuve 5.6

Soit $a = f_0,f_1,f_2,\ldots,f_n,z = f_{n+1}$ un chemin minimal dans le graphe de script. Si $f_0,f_1,f_2,\ldots,f_n,f_{n+1}$ n'est pas un chemin minimal dans le graphe de scripts valides, alors il existe nécessairement un facteur g et un indice i appartenant à $1..n$ tels que $f_i <_o g <_o f_{i+1}$ et donc $(f_1,f_2,\ldots,f_n)$ n'est pas un script valide. □

La réduction de taille entre le graphe de scripts et le graphe de scripts valides est significative. Considérons par exemple un ensemble de facteurs valides tel que tous les couples de facteurs tels que $f <_s g$ vérifient la relation $<_o$. Le graphe de scripts comportait $\frac{n^2+3n+2}{2}$ arcs alors que le graphe de scripts valides en comporte $n+1$ (voir figure 5.8). Ce cas correspondait au pire des cas pour le graphe de scripts et correspond au meilleur des cas pour le graphe de scripts valides. Si, au contraire, on envisage un ensemble de facteurs où tous se chevauchent deux à deux, alors le nombre d'arcs reste $2n$.

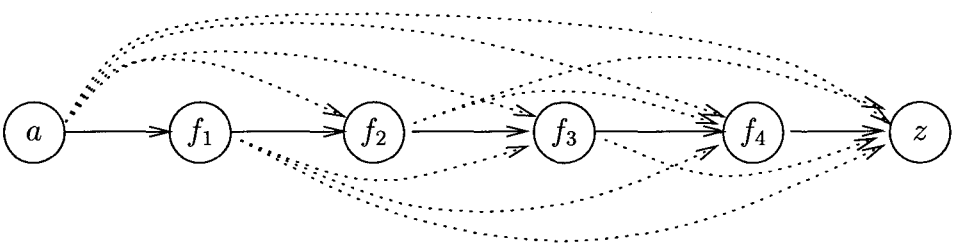


FIG. 5.8 – *Un graphe de scripts valides pour un ensemble de facteurs valides tel que tous les couples de facteurs tels que* $f <_s g$ *vérifient la relation* $<_o$. *Les arcs supplémentaires du graphe de scripts sont en pointillés.*

Prenons un exemple où nous supposons que l'ensemble des facteurs est constitué de p sous-ensembles chevauchant contenant chacun q facteurs. On considère que les facteurs de chaque sous-ensemble chevauchant se chevauchent tous deux à deux (voir figure 5.9). Dans le cas du graphe de scripts, le nombre d'arcs sera en $O(p^2q^2)$ alors qu'il sera en $O(pq^2)$ pour le graphe

de scripts valides. Dans le premier cas (graphe de scripts), un facteur donné d'un sous-ensemble chevauchant sera relié à tous les facteurs de tous les sous-ensembles chevauchant suivants. Il y aura donc q^2 arcs reliant deux sous-ensembles chevauchant, soit

$$pq + 1 + \sum_{i=1} p(p-i)q^2 = \frac{p^2q^2 - pq^2 + 2pq + 2}{2}$$

arcs au total (le terme $pq + 1$ provient de la liaison entre a et tous les autres arcs, y compris z). Dans le second cas (graphe de scripts valides), un facteur donné d'un sous-ensemble chevauchant sera relié à tous les facteurs du seul sous-ensemble chevauchant qui le suit. q^2 arcs relieront donc les sous-ensembles chevauchant entre eux, soit au total

$$2q + \sum_{i=1}^{p} q^2 = pq^2 + 2q$$

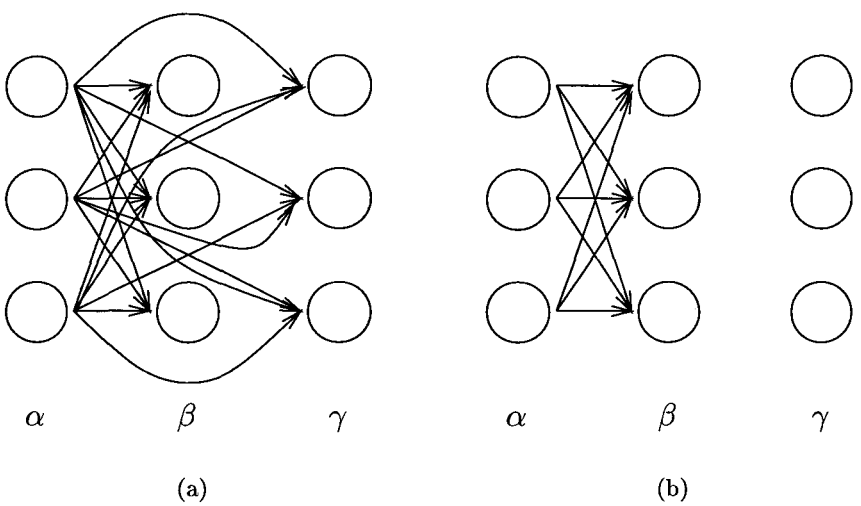


FIG. 5.9 – *Représentation des arcs reliant des sous-ensembles chevauchant possédant le même nombre de facteurs et tels que tous les facteurs se chevauchent deux à deux. Liaison entre le sous-ensemble α et les sous-ensembles β et γ : (a) pour le graphe de scripts, (b) pour le graphe de scripts valides.*

Construction

Nous nous intéressons maintenant à la construction du graphe de scripts valides. Un processus brutal consiste à visiter la liste des sommets déjà insérés dans le graphe chaque fois que nous désirons en ajouter un nouveau. En admettant que nous procédions par ajout des facteurs dans l'ordre $<_s$, la construction du graphe sera en $O(n^2)$ en temps, si n est le nombre de facteurs. C'est la construction du graphe de scripts.

Il est possible d'accélérer le processus de construction en limitant l'ensemble des sommets à visiter. Grâce à la propriété des facteurs valides, nous savons qu'un facteur doit être inclus dans un script dès qu'il peut l'être. Cela implique que si nous avons trois facteurs tels que $f <_o g <_o h$,

il n'est pas nécessaire de disposer dans le graphe de scripts valides d'un arc entre les sommets représentant f et h.

Nous indiquons une propriété permettant d'avoir un algorithme de construction incrémental du graphe. Les facteurs sont considérés dans l'ordre $<_s$. Pour chaque nouveau facteur ajouté, nous montrons qu'il est suffisant de considérer les facteurs dans les deux derniers niveaux du graphe en cours de construction pour s'assurer de la création de tous les arcs aboutissant au nouveau facteur. Nous limitons ainsi l'ensemble des sommets pouvant former un arc avec le nouveau facteur à $\{f \in V, f \in \text{pred}(z) \cup \text{pred}(\text{pred}(z))\}$. La complexité dans le pire des cas n'est pas modifiée car il peut toujours exister un ensemble où les facteurs se chevauchent deux à deux. Dans une telle situation, il est nécessaire d'inspecter tous les facteurs déjà inclus dans le graphe pour en ajouter un nouveau. Cependant une telle situation est rare et l'amélioration que nous proposons permet en pratique d'obtenir un réel gain (voir chapitre suivant).

Lemme 5.8 *Soit $G_v = (V,E,w)$ un graphe de scripts valides basé sur un ensemble de facteurs valides F_v. Soit g un facteur tel que pour tout f appartenant à $V - \{z\}$, $f <_s g$. Soit $P = \{f \in V$ tels que $succ(f) = z$ ou $succ(succ(f)) = z\}$, alors l'ensemble $\{f,(f,g) \in E\}$ est inclus dans P.*

Preuve 5.8

Raisonnons par l'absurde. Supposons qu'il existe $f \in V - \{P \cup \{z\}\}$ tel que (f,g) appartienne à E. Alors g appartient aux successeurs de f. Rappelons que f n'appartient pas à P, par conséquent il existe $h \in \text{succ}(f)$ et $h' \in \text{succ}(h)$ tels que $f <_o h <_o h' <_o z$. Mais g vérifie la relation $<_s$ pour tout facteur de $V - \{z\}$, et donc $f <_o h <_o h' <_s g$. Et donc, il existe un facteur h tel que $f <_o h <_o g$. Par conséquent, f ne peut pas être un successeur de f, (f,g) n'est pas un arc du graphe. □

Proposition 5.7 *Soit $G_v = (V,E,w)$ un graphe de scripts valides basé sur un ensemble de facteurs valides F_v. Soit g un facteur tel que pour tout f appartenant à $V - \{z\}$, $f <_s g$. Pour tout $f \in pred(z)$:*

- *si $f <_o g$ alors (f,g) appartient à E et (f,z) doit être supprimé,*
- *si $f \not<_o g$, pour tout $f_p \in pred(f)$, $(f_p,g) \in E$ si $g \notin succ(f_p)$,*
- *(g,z) appartient à E.*

Preuve 5.7

i Montrons que si il existe f tel que (f,g) appartient à E, (f,g) est effectivement ajouté à E lors de la construction. Grâce au lemme 5.8, nous savons que l'ensemble des possibles prédécesseurs de g est réduit à $\{f \in V$ tels que $\text{succ}(f) = Z$ ou $\text{succ}(\text{succ}(f))\}$. Supposons qu'il existe $f \in \text{pred}(z)$ tel que $g \in \text{succ}(f)$, alors $f <_o g$ et (f,g) est ajouté à l'ensemble des arcs (premier point). Supposons maintenant qu'il existe $f \in \text{pred}(\text{pred}(z))$ tel que $g \in \text{succ}(f)$, alors il existe $h \in \text{succ}(f)$ tel que $h <_s g$ et $h \not<_o g$, et donc (f,g) est ajouté à l'ensemble des arcs (second point),

ii (g,z) appartient effectivement à E et est toujours ajouté à E (troisième point),

iii Montrons que la construction n'ajoute pas d'arcs inutiles. Supposons qu'il existe $f \in \text{pred}(z)$ tel que $f <_o g$, seul l'arc (f,g) est ajouté. g est bien un successeur de f : comme f est un prédécesseur de z, il n'existe pas de facteur h tel que $f <_o h <_o g$. Supposons maintenant qu'il existe $f_p \in \text{pred}(\text{pred}(z))$ et qu'il n'existe pas de successeur f de f_p tel que $f <_o g$, alors seul (f_p,g) est ajouté. Cela implique qu'il n'existe pas f tel que $f_p <_o f <_o g$ et donc g est bien un successeur de f_p.

□

De cette proposition il est facile de déduire un algorithme de construction du graphe (voir algorithme 5.5). L'application de l'algorithme des plus courts chemins vu en début de chapitre permet un calcul direct de la solution optimale à partir du graphe.

ALG 5.5 Ajoute un facteur valide à un graphe de scripts valides

Entrée: $G_v = (V,E,w)$ est un graphe de scripts valides, g est un facteur valide tel que pour tout $f \in V - \{z\}$, $f <_s g$

Sortie: $G'_v = (V',E',w')$ est un graphe de scripts valide construit sur $V \cup \{g\}$

```
V' ← V ∪ {g}
E' ← E
pour tout f ∈ pred(z) faire
  si f <_o g alors
    ajouter (f,g) à E'
    enlever (f,z) de E'
  sinon {f <_s g et f ≮_o g}
    pour tout f_p ∈ pred(f) faire
      si g ∉ succ(f_p) alors
        {si l'arc (f_p,g) n'a pas déjà été créé}
        ajouter (f_p,g) à E
      fin si
    fin pour
  fin si
fin pour
ajouter (g,z) à E'
```

Résolution directe

Nous nous intéressons maintenant à la résolution directe (comme dans le cas du graphe de scripts et du graphe de scripts compact) du calcul du script valide de poids minimal à partir de l'ensemble des facteurs. Le but est évidemment de tirer parti des facteurs valides et donc de limiter l'ensemble des facteurs observés pour un facteur f donné à l'ensemble $\text{pred}(z) \cup \text{pred}(\text{pred}(z))$.

En fait, nous avons déjà remarqué plus haut qu'il n'existe des arcs qu'entre des sous-ensembles chevauchant qui se succèdent (voir figure 5.9). Nous formalisons cela et proposons un algorithme

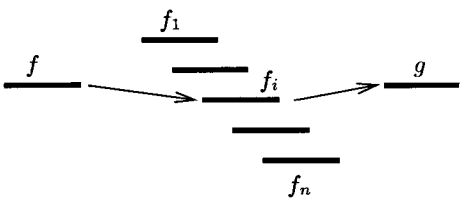
de calcul de la distance.

Définition 5.20 *Soit $F = (f_1,f_2,\dots,f_n)$ et $G = (g_1,g_2,\dots,g_m)$ deux sous-ensembles chevauchants, on dira que F **précède** G si $f_n <_o g_1$.*

Le lemme suivant montre qu'il n'existe pas d'arc entre un facteur qui précède un sous-ensemble chevauchant et un facteur qui lui succède.

Lemme 5.9 *Soit $(f_1,f_2,\dots,f_n)$ un sous-ensemble chevauchant de facteurs valides. Soit g un facteur valide tel que $f_n <_o g$, alors il n'existe pas d'arc entre pred$(\{f_1,f_2,\dots,f_n\})$ et g.*

Preuve 5.9



Soit $f \in \text{pred}(\{f_1,f_2,\dots,f_n\})$, il existe nécessairement $i \in [1..n]$ tel que $f <_o f_i$. Or on a que $f_i <_s f_n <_o g$ et donc :

$$f <_o f_i <_s f_n <_o g$$

Cela rend impossible l'existence d'un arc entre f et g (d'après la définition du graphe de scripts valides). □

Corollaire 5.1 *Soit F,G,H trois sous-ensembles chevauchant tels que F précède G et G précède H. Il n'existe pas d'arcs joignant des facteurs de F et H.*

L'algorithme de calcul du script optimal peut facilement tirer parti de ce corollaire. Il consiste en une itération du processus suivant :

- identifier un nouveau sous-ensemble chevauchant,
- relâcher les arcs existants à l'intérieur du sous-ensemble chevauchant et ne conserver que les facteurs qui ne sont pas source d'un arc,
- relâcher tous les arcs entre le sous-ensemble chevauchant précédent et le nouveau,
- faire de l'ensemble du point 2 le sous-ensemble chevauchant précédent.

Ce processus est schématisé avec la figure 5.10. L'algorithme 5.6 présente la résolution du calcul du script valide optimal basé sur cette méthode.

Proposition 5.8 *Soit F_v un ensemble de facteurs valides, l'algorithme* MEILLEURSCRIPTVALIDE *calcule le script optimal et son poids.*

Remarque 5.5 *Il est possible de proposer une heuristique plus forte encore en considérant un graphe de scripts valides basé uniquement sur l'ensemble des facteurs maximaux. L'approximation de la distance obtenue alors est assez mauvaise, on atteint assez rapidement les 10% d'erreur.*

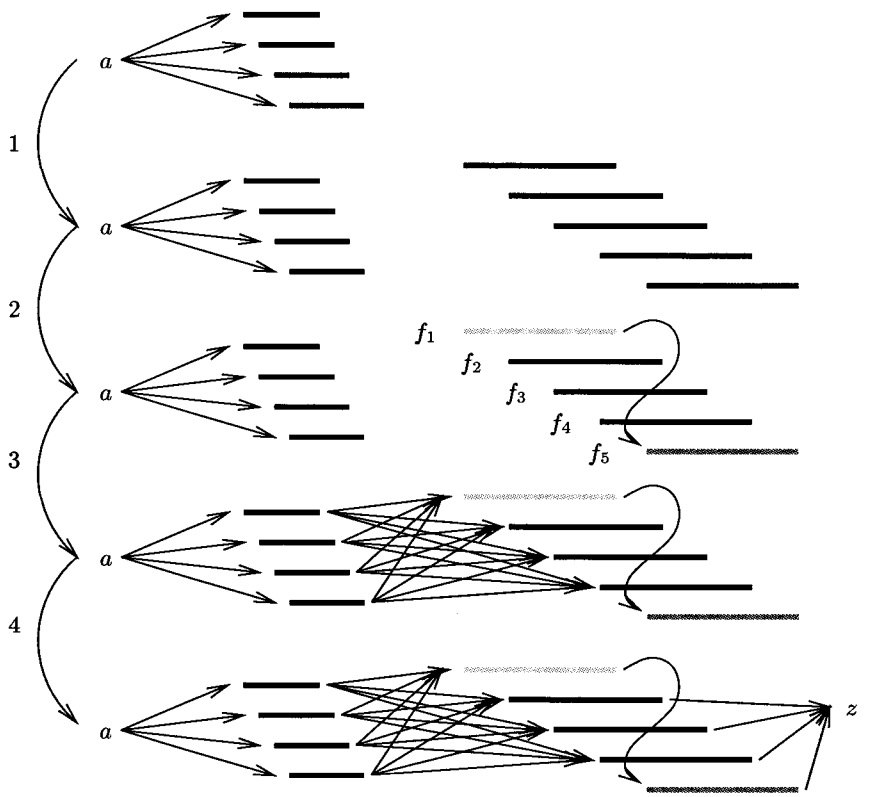


FIG. 5.10 – *Illustration de l'algorithme de calcul du meilleur script valide. Avant l'étape 1 il existe déjà un sous-ensemble chevauchant dans le graphe. La première étape consiste à identifier le sous-ensemble chevauchant suivant. La seconde étape consiste à calculer les arcs internes du nouveau sous-ensemble. On ne conserve pour la suite que les facteurs n'étant pas la source d'un arc interne : tous les facteurs hormis celui du haut,* $P = \{f_2,f_3,f_4,f_5\}$. *On ne va relier les facteurs du sous-ensemble précédent qu'avec les facteurs qui ne sont pas cibles d'un arc interne : tous les facteurs hormis celui du bas,* $Q = \{f_1,f_2,f_3,f_4\}$. *Dans la troisième étape, on crée tous les arcs entre le nouveau sous-ensemble chevauchant et son prédécesseur : tous les facteurs du sous-ensemble précédent (sauf ceux ayant un arc interne) sont reliés à tous les facteurs du nouveau sous-ensemble. L'étape suivante consiste en la réitération du processus, le sous-ensemble précédent ne contient que les facteurs de* P.

ALG 5.6 MEILLEURSCRIPTVALIDE

Entrée: F_v un ensemble de facteurs valides

Sortie: le calcul du script valide optimal

```
{CC représente le sous-ensemble chevauchant précédent}
CC ← ∅
{C représente le nouveau sous-ensemble chevauchant}
C ← ∅
f ← a
tant que il existe des facteurs non visités faire
  { - 1 - calcul du nouveau sous-ensemble chevauchant}
  g ← f
  tant que f ≮_o g faire
    C ← C ∪ {g}
    f ← g
    g ← le facteur suivant
  fin tant que
  { - 2 - calcul des arcs internes}
  {P représente les facteurs source d'un arc}
  P ← C
  {Q représente les facteurs but d'un arc}
  Q ← C
  pour tout u ∈ C faire
    pour tout v ∈ C tel que u <_o v faire
      RELÂCHER(u,v)
      P ← P − {u}
      Q ← Q − {v}
    fin pour
  fin pour
  { - 3 - relâchement avec le sous-ensemble précédent}
  pour tout u ∈ CC faire
    pour tout v ∈ Q faire
      RELÂCHER(u,v)
    fin pour
  fin pour
  CC ← P
fin tant que
retourner w(z)
```

5.4 Résumé

Nous avons d'abord vu que la donnée d'un script ou d'une combinaison de facteurs, un f-**script**, étaient équivalents. De cette proprité nous avons déduit que la recherche du script de poids le plus faible pouvait être résolue grâce à une représentation de tous les f-scripts dans un **graphe de scripts**.

L'inconvénient du graphe de scripts est d'une part qu'il comporte un grand nombre d'arcs et d'autre part qu'il nécessite de disposer de l'ensemble de tous les facteurs (pas seulement les maximaux). Le premier point accroît le temps de calcul alors que le second accroît l'espace mémoire. Nous avons défini le **graphe de scripts compact** qui se base sur des propritétés vérifiées par les fonctions de poids telles que seuls les facteurs maximaux et un petit nombre de sous-facteurs nécessitent d'être inclus dans le graphe. Cela a pour conséquence de diminuer le nombre d'arcs. D'autres propriétés nous ont également permis de ne pas inclure certains arcs. La mise en avant de ces propriétés a permis d'aboutir à un algorithme plus rapide pour le calcul de la distance de transformation. Si les propriétés ne sont pas vérifiées, l'algorithme est heuristique ; en pratique nous avons constaté que cette heuristique est bonne (voir chapitre suivant).

Nous avons proposé une autre réduction possible du graphe : le **graphe de scripts valides**. Dans ce cadre, on considére que si un facteur commun entre les deux séquences est disponible, alors ce facteur apporte de la connaissance et doit être inclus dans un script si cela est possible. Cela revient à considérer que le coût de copie d'un facteur est toujours inférieur au coût d'insertion.

Dans le chapitre suivant nous comparons les résultats obtenus par ces algorithmes et détaillons les conditions qui sont vérifiées par les fonctions de poids.

Chapitre 6

Discussion de la mise en œuvre de la distance de transformation

Nous avons proposé au chapitre 4 la définition d'un script de transformation et de la mesure qui lui était associée. Cette définition est plutôt une définition de famille de distances en ce sens que les différents ensembles d'opérations, les systèmes générateurs, et les poids associés à ces opérations permettent de définir une grande variété de distances.

Nous nous sommes intéressés dans ce travail à une première approche au travers du système générateur de reconstruction $\{ins,cop\}$. Au chapitre précédent, nous avons proposé un algorithme permettant le calcul de la distance associée à ce système générateur.

Ce chapitre décrit des réflexions et les choix qui ont été faits lorsque nous avons voulu obtenir une "vraie" distance, c'est-à-dire avoir un résultat numérique.

Un premier problème fût celui du calcul de l'ensemble des facteurs communs. Il se pose le problème de la définition de ces facteurs. Nous nous sommes longtemps contentés de considérer des facteurs exacts. Cela présentait le double avantage de la rapidité de calcul et la non ambiguïté de la définition. Mais ce choix n'était pas satisfaisant. En se limitant à des facteurs exacts, nous ne détections pas des grands facteurs à cause de quelques mutations. Nous nous sommes alors orientés vers des facteurs ayant une structure assez rigide mais qui semble donner des résultats corrects. Nous présentons l'algorithme de calcul de ces facteurs. Mais considérer l'ensemble de tous les facteurs communs n'est pas envisageable, même avec le graphe de scripts compact. Nous étudions donc une borne de longueur minimale des facteurs.

Le problème suivant est le choix des fonctions de poids pour les opérations d'insertion et de copie (section 6.2). Nous avons vu au chapitre 3 combien une méthode de compression était le reflet d'un modèle défini par le schéma de codage. Il en va de même pour le choix des fonctions de poids. Il a fallu définir celles-ci en prenant garde à ne pas favoriser arbitrairement certaines configurations de facteurs plutôt que d'autres tout en essayant d'avantager au mieux ce que nous pensons être un bon critère de ressemblance. Le but étant de rester le plus proche de l'objectivité pour ne pas perdre cette bonne propriété de la distance informationnelle dès lors qu'on ne peut en avoir qu'une approximation. Nous nous interrogeons ensuite sur le codage des items des fonctions de poids, puis sur les propriétés de la distance. Celle-ci n'est clairement pas symétrique. Ce que nous mesurons avec la distance de transformation est la quantité d'information qu'il est nécessaire d'ajouter à S pour passer de S à T. Rien ne permet de dire qu'il faut ajouter la même quantité

d'information pour passer de S à T que de T à S. De même, il est possible de construire des cas pour lesquels l'inégalité triangulaire n'est pas vérifiée. Cependant, sur les échantillons que nous étudions, l'inégalité triangulaire est vérifiée.

Dans une dernière partie (section 6.3), nous comparons les performances des trois méthodes proposées au chapitre précédent. Dans un premier temps nous regardons si les conditions nécessaires sont vérifiées, puis comparons expérimentalement les temps de calcul et les erreurs introduites par le graphe de scripts compact et le graphe de scripts valides.

Les tests que nous effectuons dans les deux dernières sections utilisent divers ensembles de séquences, que nous appelons échantillons et présentons maintenant.

Échantillon 1 *Il s'agit de 20 séquences du gène du cytochrome b chez divers cétacés, numéros d'accession X92524 à X92543. Les séquences sont toutes de longueur 1140pb.*

Échantillon 2 *9 séquences d'ARN ribosomique 18S de truffe, numéros d'accession AF132501 à AF132509. La longueur des séquences varie entre 585pb et 876pb.*

Échantillon 3 *Deux séquences de génomes mitochondriaux. Le premier génome est celui d'un alligator, numéro d'accession Y13113, de longueur 16646pb et le second est celui d'un opossum, numéro d'accession Z29573 de longueur 17084pb.*

Échantillon 4 *Un ensemble de 46 séquences d'ARN ribosomal de longueur variant entre 300pb et 400pb. Plus de détails sur ces données pourront être trouvés au chapitre suivant.*

6.1 Choix des facteurs communs

Nous présentons dans cette section d'une part l'algorithme de recherche de l'ensemble des facteurs communs (appelés dans la suite simplement facteurs) que nous utilisons pour le calcul de la distance de transformation et d'autre part des considérations relatives à la borne minimale pour la longueur des facteurs.

6.1.1 Recherche d'un ensemble de facteurs

Lorsque nous avons voulu mettre en œuvre le calcul de la distance, nous nous sommes retrouvés confrontés au problème de la définition d'un facteur. Plus précisément, nous devions définir un critère permettant de dire que deux segments sont suffisamment similaires pour former un facteur commun. Comme nous l'avons dit précédemment, dans un premier temps nous nous sommes limités à ne considérer que des facteurs exacts. Il existe divers algorithmes permettant de les calculer [CR94]. Nous avons choisi l'algorithme de Leung et *al.* [LBBK91]. Cet algorithme met en œuvre une table de hachage de blocs identiques d'une longueur donnée associée à une table de liens de ces blocs. Les facteurs sont obtenus par extension des blocs grâce à la table de liens.

L'avantage de cet algorithme est qu'il est facilement extensible à la détection de facteurs avec erreurs. Les segments communs ont alors une structure particulière : chaque segment est un agrégat de blocs identiques d'une longueur minimale donnée, séparés par des blocs qui contiennent

des erreurs (substitutions et insertions/délétions) d'une longueur maximale donnée, que nous nommons blocs d'erreur.

Facteurs exacts

Leung et *al.* [LBBK91] ont proposé cet algorithme basé sur le principe d'une table de hachage qui permet l'identification rapide de petits segments communs identiques. Ces occurrences sont liées entre-elles et la recherche de facteurs plus grands est effectuée par extension de ces petits blocs.

Les blocs identiques doivent avoir une taille supérieure à un paramètre b et les blocs d'erreur une taille inférieure à un paramètre e. De plus, on impose que les deux segments d'une paire identifiée commune partagent un super-bloc, c'est-à-dire un bloc identique de longueur suffisamment grande, de taille supérieure à un paramètre c. L'algorithme étant dédié à la recherche dans plusieurs séquences, une contrainte supplémentaire de multiplicité choisie par l'utilisateur, correspondant au nombre d'occurrences d'un agrégat parmi les différentes séquences, est ajoutée.

Puisque nous nous intéressons à la recherche d'un ensemble de facteurs communs entre deux séquences, nous avons supprimé la contrainte de super-bloc et de multiplicité dans notre variante.

Calcul des blocs. La première étape consiste à calculer l'ensemble des blocs identiques de taille b et de construire une table liant ces blocs. Nous appelons ces blocs des **b-blocs**.

Ce calcul est effectué sur la séquence composite : la séquence résultant de la concaténation des deux séquences S et T. Chaque b-bloc est représenté par en entier, appelé **clé**. La première étape consiste donc à remplacer la séquence composite par une séquence de $N - b + 1$ entiers (N étant la longueur de la séquence composite). Le calcul de la clé se fait de manière tout à fait classique : si $c_b c_{b-1} \dots c_1$ est un b-bloc écrit sur un alphabet Σ, la clé correspondante sera

$$\sum_{i=1}^{b} \text{code}(c_i)|\Sigma|^{i-1}$$

où code(c_i) est une application bijective qui associe au symbole c_i un entier entre 0 et $|\Sigma| - 1$ (avec $\Sigma = \{\texttt{A},\texttt{G},\texttt{C},\texttt{T}\}$, on a code(A)=0, code(G)=1, code(C)=2, code(T)=3).

Dans le même temps que la transformation de la séquence composite en une séquence d'entiers s'effectue, on met à jour une **table de liens** dont chaque case i contient la prochaine occurrence du même b-bloc débutant à la position i dans la séquence composite. Si il n'y a pas d'occurence suivante, la case reste vide.

Exemple 6.1 *Prenons deux séquences* S = AGAGGTAAGCTTTTT *et* T = AGATAATAACTTTCT *et fixons* $b = 2$, *les tableaux suivants montrent la séquence composite, son codage en une suite de clés, le tableau des liens entre les mêmes b-blocs.*

Indices :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Séquence composite :

A	G	A	T	A	A	T	A	A	C	T	T	T	C	T	A	G	A	G	G	T	A	A	G	C	T	T	T	T	T

Tableau des clés :

1	4	3	12	0	3	12	0	2	11	15	15	14	11	12	1	4	1	5	7	12	0	1	6	11	15	15	15	15	

Tableau des liens :

16	17	6	7	8		15	22		14	12	26		25	21	18		23								27	28	29		

On a : clé(AG)=code(A)×4^1+code(G)×4^0 = 0 + 1=1, clé(GA)=code(G)×4^1+code(A)×4^0 = 4 + 0=4, clé(AT)=code(A)×4^1+code(T)×4^0 = 0 + 3=3, clé(TA)=code(T)×4^1+code(A)×4^0 = 3 × 4 + 0=12.

Le calcul des clés et des liens s'obtient facilement en une seule passe sur la séquence composite. Le calcul de la clé associée à un b-bloc est très efficace en remarquant que la clé du b-bloc en position $i+1$ se déduit de celle en position i grâce à un décalage binaire et un ou exclusif.

Il faut cependant noter que b ne peut pas prendre toutes les valeurs. En effet, pour construire le tableau des liens il est nécessaire de disposer d'un tableau indicé sur les clés, soit de taille $b^{|\Sigma|}$. Les limites de la mémoire d'une machine classique sont vite atteintes et des valeurs de b supérieures à 15 ne sont pas envisageables pour l'alphabet des nucléotides.

Extension des b-blocs sans erreurs. A partir du tableau des liens il est facile de calculer des blocs identiques de taille supérieure ou égale à b. Soit i un indice dans la séquence composite tel que $i < |S|$. Le tableau des liens contient à la position i la position $j > i$ de la prochaine occurrence du b-bloc $S[i..i+b-1]$ dans la séquence composite. Prenons un entier j tel que $j > |S|$. Pour savoir si il existe un segment commun identique de taille supérieure à b entre les positions i et j, il suffit de regarder si la clé à l'indice $i+b$ est la même que celle à l'indice $j+b$.

Exemple 6.2 *Avec l'exemple précédent, si l'on prend $i = 10$, l'occurrence suivante, donnée par la valeur du tableau de liens à l'indice i, est à l'indice $j = 14$. Cette valeur est inférieure à la longueur de S, on passe donc au j suivant, égal à 25. Le b-bloc à l'indice $i+b = 10+2 = 12$ porte la clé 15, le b-bloc à l'indice $j+b = 25+2 = 27$ porte également la clé 15. Il y a donc un segment identique (CTTT) de longueur 4 entre les positions 10 (dans T) et $25-15 = 10$ (dans S) entre S et T.*

Si maintenant on ne trouve pas la même clé entre les indices $i+b$ et $j+b$, on sait alors qu'il y a au moins un des symboles qui diffère entre les b-blocs $S[i+b..i+2b-1]$ et $T[j+b-|S|..j+2b-1-|S|]$. On procède alors à l'extension du bloc identique position par position au lieu de le faire b positions par b positions. Il y a donc au maximum $b-1$ comparaisons de clés à effectuer.

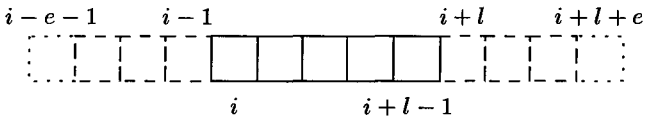
Exemple 6.3 *Continuons sur le même exemple. Nous avons maintenant identifié un b-bloc identique entre $i+b$ et $j+b$. On s'intéresse donc maintenant aux positions $i+2b$ et $j+2b$. Les clés à ces indices sont respectivement 11 et 15. Il faut donc comparer les clés aux indices $i+b+1$,$j+b+1$ (13,28) ; $i+b+2$,$j+b+2$ (14,29) ; ... ; $i+2b-1$,$j+2b-1$. Dans le cas présent, les clés aux indices $i+b+1 = 13$ et $j+b = 1 = 28$ sont différentes. On ne peut donc pas trouver de segment identique commun de longueur supérieure à 4 débutant en position 10 dans S et en position 10 dans T.*

Lorsque l'on cherche à étendre les blocs, on cherche en fait à calculer l'ensemble des facteurs maximaux communs entre S et T. Les sous-facteurs peuvent facilement être obtenus par déduction.

Facteurs avec erreurs

Le processus de calcul de segments communs avec erreurs est similaire à celui précédemment exposé. Nous avons vu que dans le cadre dans lequel nous travaillons ici, un segment est composé de plusieurs blocs identiques séparés par des petits blocs d'erreurs. Notons e la longueur maximale des blocs d'erreurs. En limitant les erreurs autorisées à des substitutions, nous aboutissons à un algorithme plus simple que celui autorisant également les insertions/délétions.

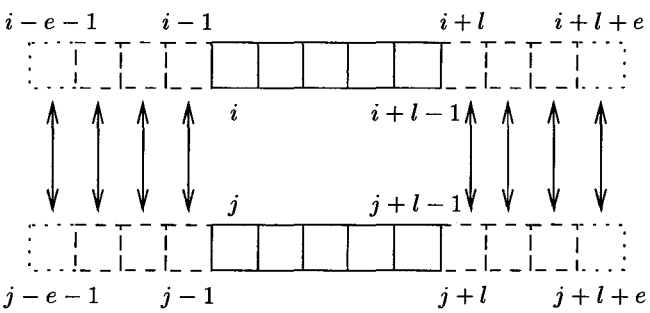
Extension des b-blocs avec substitutions. Imaginons que nous ayons deux blocs identiques à des positions respectives i et j et de longueur l (ces blocs sont obtenus par le processus décrit dans le paragraphe précédent). Notre problème est maintenant de trouver une extension à ces blocs pour rechercher deux segments communs plus grands. Nous avons autorisé e erreurs, par conséquent les seules extensions possibles sont de trouver deux blocs identiques (de longueur supérieure ou égale à b) débutant dans les intervalles $i+l..i+l+e$ et $j+l..j+l+e$ pour l'extension par la droite et se terminant dans les intervalles $i-e-1..i-1$ et $j-e-1..j-1$ pour l'extension par la gauche :



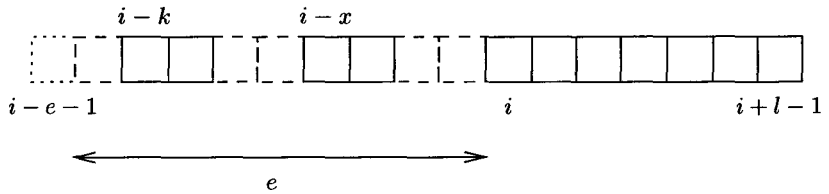
Pour être certain d'envisager tous les facteurs possibles, il est nécessaire d'examiner toutes les positions à gauche et à droite pour lesquelles un bloc identique commun est trouvé dans ces intervalles.

Dans la suite nous nous intéressons uniquement à l'extension gauche, l'extension droite est obtenue de manière similaire.

Si nous n'autorisons que des substitutions, l'écart entre les blocs identiques sera le même dans les deux séquences. Les occurrences dans S et T du bloc identique que nous cherchons ne peuvent avoir que le même écart par rapport aux occurrences dans S et T du bloc identique que nous étendons. Cela signifie que nous devons chercher si il existe un bloc identique commun uniquement pour les couples de positions $(i-1,j-1)$, $(i-2,j-2)$, ... , $(i-e-1,j-e-1)$ pour l'extension à gauche :



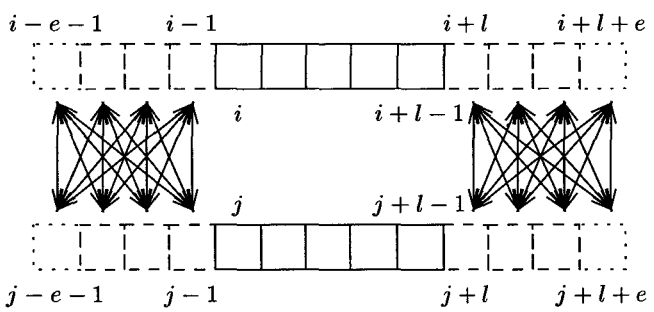
De plus, si il existe un bloc identique commun aux positions $i-k$ et $j-k$, $k \in 1..e+1$, alors celui-ci conduit à un segment commun plus long que tout bloc identique communs entre les positions $i-x$ et $j-x$, $x \in 1..k-1$. En effet, si l'on suppose qu'il existe un bloc identique commun aux positions $(i-x,j-x)$ et un bloc identique commun aux positions $(i-k,j-k)$ alors l'extension du segment commun $S[i-x..i+l-1],T[j-x,j+l-1]$ aboutira nécessairement au segment commun $S[i-k..i+l-1],T[j-k,j+l-1]$ puisqu'il y aura moins de e erreurs entre les blocs identiques en positions $(i-x,j-x)$ et $(i-k,j-k)$.



Par conséquent, la recherche d'un nouveau bloc identique doit débuter aux positions les plus éloignées, $(i-e-1,j-e-1)$, correspondant au nombre d'erreurs maximal. L'extension s'arrête dès que l'on trouve un bloc identique. Ensuite il suffit de compter le nombre réel d'erreurs.

Comme e est la plupart du temps choisi petit, le bloc identique débutant à la position x est souvent un bloc identique non étendu dont l'extension aboutirait au bloc identique débutant en position k. Si l'on commence par calculer l'ensemble des blocs identiques maximaux, leur extension avec substitutions est une accélération de l'algorithme de base.

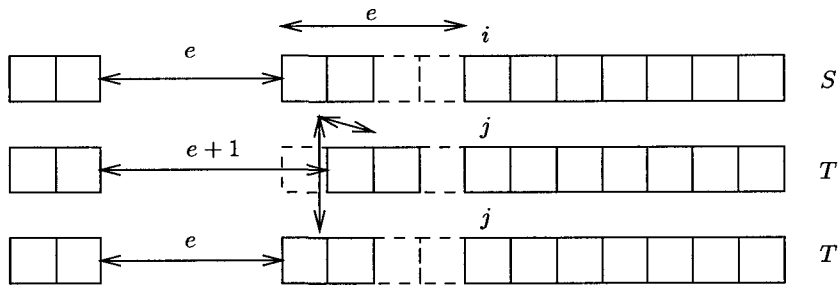
Ajout des insertions/délétions. Si nous voulons prendre en compte les insertions/délétions, il est nécessaire cette fois d'envisager l'ensemble des couples $i-e-1..i-1$ et $j-e-1..j-1$.



Comme dans le cas sans insertions/délétions, on recherche les blocs identiques aux positions les plus éloignées du bloc courant.

Cela n'empêche pas de devoir considérer un plus grand nombre de couples puisque le fait d'autoriser les insertions/délétions fait que la progression n'est pas forcément parallèle. Le schéma ci-dessous montre qu'il est indispensable d'envisager toutes les combinaisons si l'on veut être certain de trouver l'ensemble des facteurs répondant à la description que nous en avons donnée : tous les couples de segments construits comme des successions de blocs identiques de longueur

supérieure ou égale à b séparés par des blocs d'erreurs de longueur inférieure ou égale à e.



6.1.2 La longueur minimale des facteurs

D'un côté, pour des questions de temps de calcul, il n'est pas possible de conserver l'ensemble de tous les facteurs communs à deux séquences. Il faut donc effectuer une préselection avant de construire le graphe. D'autre part, quelle est la signification d'un facteur commun de trois bases pour deux séquences de longueur 1000? Il paraît naturel de penser qu'il n'a pas de sens. Nous n'avons pas un avis aussi catégorique. Certes affirmer que nous avons repéré une similitude en reconnaissant un facteur court n'est pas fondé, mais, si ce facteur court est entouré de facteurs plus longs, et donc plus significatifs, l'importance que nous devons accorder au facteur court repéré s'en trouve accrue.

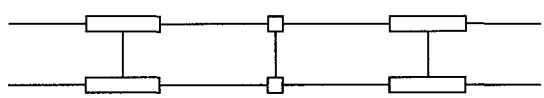
Nous proposons ici de déterminer un critère de **longueur minimale des facteurs** (**MFL** en abrégé).

Une borne expérimentale?

Commençons par observer le comportement de la distance en fonction de la longueur minimale des facteurs. Le calcul de la distance de transformation correspond à la recherche d'un minimum. On peut donc s'attendre à ce que la valeur de la distance atteigne un minimum et que l'ajout de facteurs petits ne permette pas d'améliorer la distance. Nous savons que le calcul de la distance est discriminant : l'ajout de facteurs ne se fait que si leur prise en compte diminue la valeur de la distance (sauf dans le cas du graphe de scripts valides si il est utilisé comme une heuristique). Nous pouvons alors nous demander si d'une manière générale, les petits facteurs ne sont jamais intéressants.

La figure 6.1 représente cette évolution pour trois séquences extraites de l'échantillon 4 (voir page 124). On constate que la distance diminue sans cesse, même avec l'ajout de petits facteurs. Comment expliquer cela?

Nous avons déjà exprimé dans le chapitre précédent que nous avions un choix plus complexe que celui de copier un segment ou de l'insérer. Il faut en effet se rappeler que c'est la suite d'opérations `insertion - copie - insertion` qui est remplacée par une seule opération d'`insertion`. Cela implique que la validité d'un facteur dépend largement du contexte dans lequel ce facteur est copié. Il est possible par exemple qu'un petit facteur apporte un gain d'information si il se trouve dans un bon contexte :



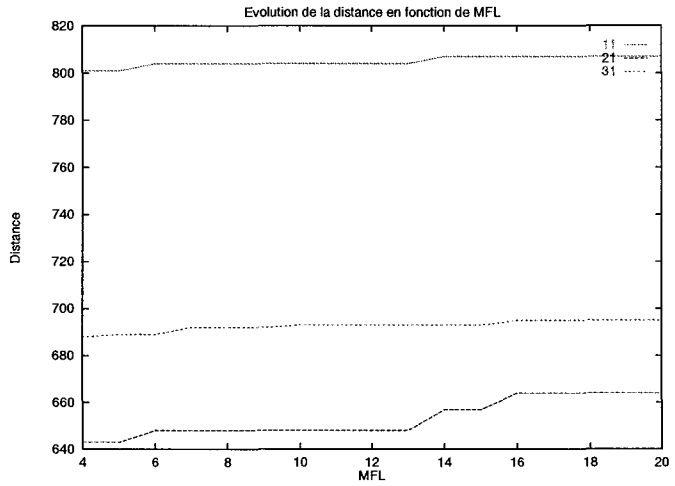
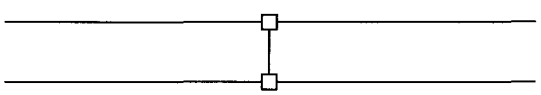


FIG. 6.1 – *Variation de la distance en fonction de MFL. Trois comparaisons de séquences de longueur 400pb sont effectuées (les facteurs calculés sont des facteurs exacts) et montrent que la distance ne cesse de décroître avec la diminution de MFL.*

alors que si ce petit facteur est seul, il n'apportera rien :



Donnons un exemple numérique pour montrer que cela est effectivement possible. Supposons avoir deux séquences de même longueur. Dans un premier schéma, on suppose que l'on a les deux scripts suivants (les déplacements sont nuls) :

Script 1		Script 2	
Opération	Longueur du segment	Opération	Longueur du segment
copie	a	copie	a
insertion	10 - x		
copie	$2x$	insertion	20
insertion	10 - x		
copie	b	copie	b

La figure 6.2 représente le comportement des coûts associés aux deux scripts. Les valeurs de a et b ne changent en rien l'abscisse à laquelle les deux courbes se croisent, pour la figure on a $a = b = 10$. La seule donnée importante ici est la valeur de x, c'est-à dire la longueur de la copie entre les segments de longueur a et b. Par conséquent, le point d'intersection des deux courbes, qui représente le **point critique** ou copie et insertion ont le même coût, **est indépendant de la longueur des séquences**. On constate que dès que la copie centrale est de longueur 8, le script 1 devient plus intéressant que le script 2. Un facteur de longueur 8 ne peut être considéré comme court que relativement à la longueur des séquences. Si nous imaginons que nos séquences sont de longueur 40000, alors effectivement 8 est un facteur court. Observons maintenant le

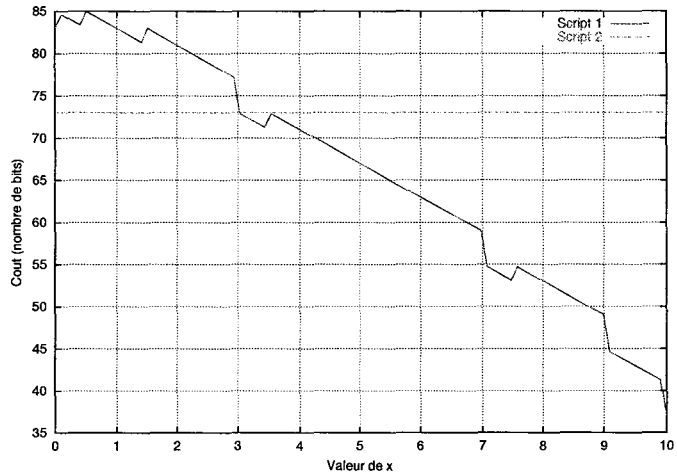


FIG. 6.2 – *Comportement des coûts des scripts 1 et 2 en fonction de la longueur de la copie centrale.*

comportement des deux scripts suivants :

Script 3		Script 4	
Opération	Longueur du segment	Opération	Longueur du segment
insertion	$20000 - x$		
copie	$2x$	insertion	40000
insertion	$20000 - x$		

La figure 6.3 représente les coûts. Cette fois, il est nécessaire que le facteur soit de longueur supérieure ou égale à 21 pour que la copie apporte un gain, c'est-à-dire que $x \geq 10.5$.

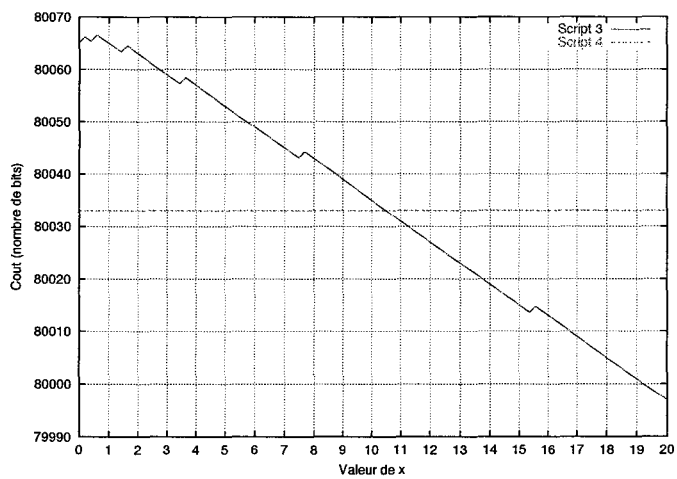


FIG. 6.3 – *Comportement des coûts des scripts 3 et 4 en fonction de la longueur de la copie centrale. Le fait que cette copie soit distante d'environ 20000 bases d'autres copies fait que la longueur à partir de laquelle il est intéressant de copier est grande : 21.*

Cet exemple numérique illustre le fait qu'il est impossible de donner une valeur par défaut à

MFL sans prendre le risque de ne pas supprimer des facteurs permettant d'aboutir à un script plus court. La prise en compte d'un facteur dépend surtout de la **densité de facteurs** qui existe autour de lui. Des exemples dans le chapitre suivant montrent que de grands facteurs peuvent être éliminés parce qu'ils sont isolés. Mais il est facile de montrer que des valeurs de MFL inférieures à 3 ne permettent jamais d'améliorer le script. Pour une copie centrale de longueur 2 telle que l'écart avec les facteurs environnant soit égal à 1 et le déplacement nul, ce qui correspond au cas le plus favorable, le poids du script est 17 alors que le poids de l'insertion remplacant la copie est 15. Cependant, il est impossible d'un point de vue pratique de considérer tous les facteurs de taille supérieure ou égale à 3. Il faut donc choisir une valeur qui ne soit pas trop pénalisante, c'est-à-dire qui ne va pas donner une distance trop éloignée de la distance minimale et qui permette un calcul praticable.

Etude du rapport gain en distance / perte de temps. Nous pouvons également regarder ce qu'il nous en coûte en temps de calcul de faire diminuer la distance. Nous nous intéressons alors au rapport entre la différence du nombre d'arcs relâchés[10] et la différence de distance pour deux valeurs successives de MFL (MFL-1 et MFL). Cela donne une mesure de l'effort relatif qui est nécessaire au gain d'un point de la distance.

La figure 6.4 présente la variation de ce rapport pour différentes valeurs de MFL pour les mêmes séquences que celles de la figure 6.1. Nous remarquons que l'effort le plus important est fourni pour le passage de MFL=5 à MFL=4, sans diminuer significativement la valeur de la distance. On dira que le **point d'effort** se situe entre 4 et 5.

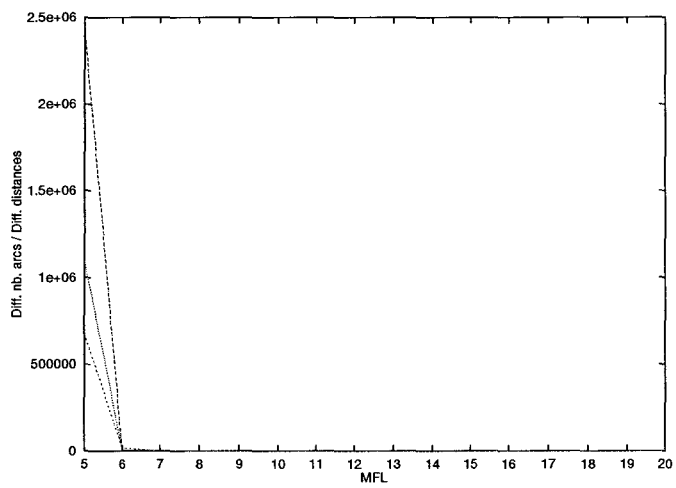


FIG. 6.4 – *Variation du rapport nombre d'arcs supplémentaires/gain en distance. Chaque point a pour coordonnées ce rapport (passage de MFL à MFL-1) et la valeur de MFL. Le point d'abscisse 5 représente le rapport pour le passage de MFL=5 à MFL=6. Les séquences sont celles de la figure 6.1. Il est nécessaire d'exécuter de nombreux calculs supplémentaires pour obtenir le script minimal avec MFL=4, le point d'effort est donc entre 5 et 4.*

Pour l'échantillon 2 (page 124), la figure 6.5 montre que le point d'effort se situe au passage de 6 à 5. La variation des distances est donnée figure 6.6. Le gain lorsque l'on passe de 6 à 5 est négligeable.

10. Nous montrons plus loin que le nombre d'arcs est proportionnel au temps d'exécution.

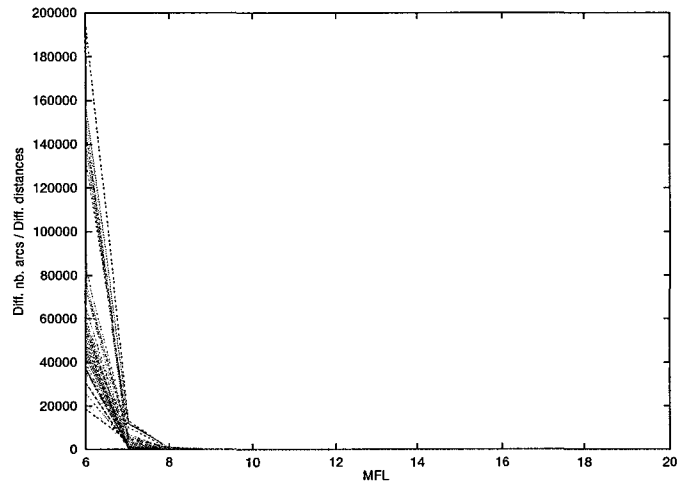


FIG. 6.5 – *Variation du rapport nombre d'arcs supplémentaires/gain en distance pour différentes valeurs de MFL. Séquences de l'échantillon 2. Cette fois l'effort doit être fait pour passer de 6 à 5.*

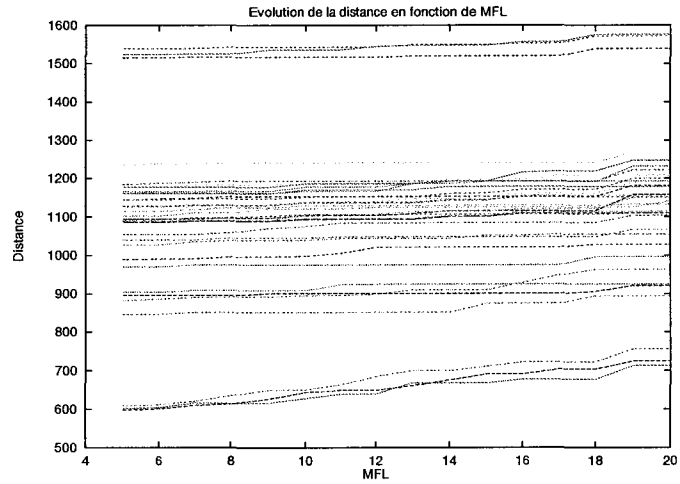


FIG. 6.6 – *Variation de la distance en fonction de MFL pour les séquences de l'échantillon 2. Le gain sur la valeur de la distance pour une valeur de MFL en deça du point d'effort est négligeable.*

On définit de manière assez abstraite le point d'effort comme la première valeur du paramètre MFL telle que les passage à MFL-1 nécessite un grand nombre de calculs supplémentaires, par exemple 1 seconde de calcul en plus par point de distance gagné.

Utiliser la valeur du point d'effort comme valeur par défaut pour MFL peut se justifier par le fait qu'une augmentation massive du temps de calcul pour un faible gain en précision, c'est-à-dire une distance plus faible, est inutile. Le problème est alors de proposer un critère de sélection du point d'effort. Pratiquement, cela pose également le problème d'estimer par avance le temps de calcul pour chaque valeur de MFL. Si une telle estimation ne paraît pas inaccessible, elle nécessite dans tous les cas de connaître le nombre de facteurs d'une taille donnée, ce qui est coûteux en temps.

Valeur théorique

Le problème de déterminer une valeur par défaut pour MFL reste entier. Nous proposons de ne pas inclure les facteurs qui peuvent potentiellement représenter une similitude due au hasard. Cette longueur de facteurs, autour de laquelle il est difficile de discerner si le facteur représente une réelle similitude, est déterminée par la valeur de la longueur moyenne d'un facteur commun entre deux séquences aléatoires. Cette valeur dépend bien sûr de la longueur des séquences. Plus les séquences sont longues, et plus il existe de chances de trouver un facteur commun long.

Si nous disposons de deux séquences de longueur l écrites sur un alphabet de taille t, alors en deçà de $log_t l$ il est plus probable que le facteur soit le résultat du hasard plutôt que d'une réelle similitude.

Pour les séquences d'ADN, on aura :

$$\text{MFL} = log_4(\text{longueur de } T)$$

Le tableau ci-dessous donne quelques valeurs d'une telle borne pour différentes longueurs de séquences :

Longueur des séquences	MFL
< 256	4
< 1024	5
< 4096	6
< 16384	7
< 65536	8

Les valeurs de longueur minimale paraissent effectivement suffisamment faibles pour nous garantir une recherche parmi un ensemble assez grand de facteurs.

Utiliser cette borne comme valeur pour le paramètre MFL ne permet pas de calcul praticable avec le graphe de scripts. Cela est facilement compréhensible : le repérage d'un facteur de taille 100 avec une valeur de MFL égale à 8 implique la création de 4371 sous-facteurs. Nous pouvons alors utiliser le graphe de scripts compact, qui permet de ne créer les sous-facteurs que lorsque cela est nécessaire et donc de préserver le temps de calcul, et de conserver une bonne approximation de la distance minimale (cela est montré plus loin dans ce chapitre).

Remarquons pour finir que les valeurs de MFL indiquées dans le tableau ci-dessus sont proches de celles des points d'effort distingués dans les figures 6.4 et 6.5 : 4 pour des séquences de longueur 400pb dans le premier cas, et 5 pour des longueurs de 600 à 800pb pour le second.

6.2 Choix d'une fonction de poids, propriétés de symétrie

Nous avons défini au chapitre 4 le concept général de la distance de transformation. Cette définition pose le cadre de la distance et il apparaît clairement qu'il est possible de dériver plusieurs variantes de la distance de transformation grâce à des définitions différentes des fonctions de poids. Partant d'une fonction naïve, nous aboutissons à une fonction satisfaisante pour la mesure de similarité que nous cherchons à définir.

Dans un second temps, nous détaillons les raisons de l'asymétrie de la distance de transformation. Nous proposons une fonction de poids qui rend la distance symétrique pour le cas où les facteurs du script ne se chevauchent pas. Cependant, celle-ci n'est pas satisfaisante d'un point de vue informationnel car elle nécessite de coder plus d'informations qu'il n'est nécessaire. Puis nous montrons que notre distance ne vérifie pas l'inégalité triangulaire, mais que sur les échantillons que nous testons, elle est vérifiée. Nous discutons ensuite du codage des items des fonctions de poids. Lorsque nous mesurons la distance entre deux facteurs nous devons coder la longueur de l'insertion entre les facteurs, la longueur de la copie, etc. Ce sont les **items** des fonctions de poids. Nous terminons cette section par une étude expérimentale des fonctions de poids et des codages définis.

Pour simplifier l'expression des fonctions dont nous allons discuter nous avons choisi de représenter les items de celles-ci par des expressions reflétant l'ordre de grandeur du codage de chaque item :

- $2\times[f,g]_x$ représente le coût associé au codage de l'insertion entre les facteurs f et g. L'indice x indique dans quelle séquence l'écart entre f et g est mesuré,
- $|x|$ représente le coût associé au codage de la copie de longueur x, de la position x dans une séquence ou du déplacement de longueur x.

6.2.1 Poids

Dans la suite, nous allons toujours définir la fonction de poids w directement, sans détailler les fonctions de coût d'insertion i et de coût de copie c. Nous allons voir que la définition de la fonction de poids influence directement le type de similitude recherchée. Rappelons que

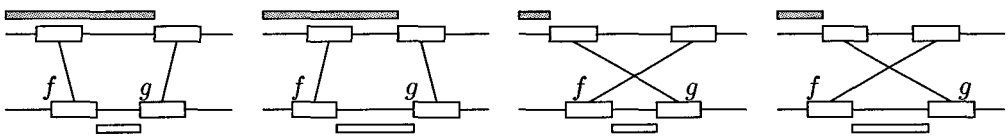
$$w(f,g) = i(f,g) + c(g)$$

Pour chaque fonction de poids envisagée, nous proposons un schéma où sont indiquées les longueurs qui sont codées (voir la figure plus loin). Sur chaque schéma, la séquence jouant le rôle de source est représenté en haut alors que la séquence jouant le rôle de cible est représentée en bas. Les facteurs sont représentés par deux segments reliés. Parmi les schémas, nous présentons un exemple où deux facteurs ne se croisent pas et un autre où il y a croisement, et pour chacun nous envisageons le cas symétrique (S devient T et T devient S). La longueur de l'insertion est représentée par un segment blanc (en bas) et la longueur du codage de la position par un segment grisé (en haut).

Des poids bruts.

Nous présentons d'abord une fonction de poids naïve. Le codage des copies se fait par référence absolue à la position dans la séquence source. La fonction de poids est alors :

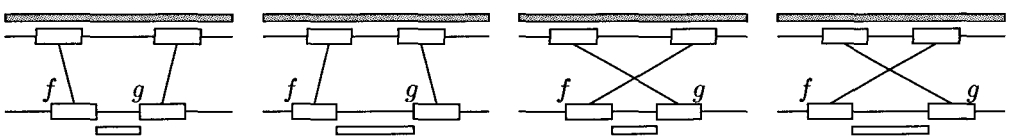
$$w_0(f,g) = 2 \times [f,g]_t + |g| + |p_g^s|$$



Il apparaît immédiatement que cette fonction de poids est peu satisfaisante. Elle favorise en effet les facteurs dont le segment dans la séquence source se trouve au début. D'autre part, on peut noter une asymétrie évidente : $d(S,T) \neq d(T,S)$.

L'excès inverse est de considérer le codage d'une position sous un format fixe:

$$w_0'(f,g) = 2 \times [f,g]_t + |g| + \text{cste}$$



On considère alors que chaque position nécessite le même poids de codage. Comme la plus grande position que l'on puisse coder sera la longueur de la séquence source, la constante devrait être égale à $|S|$. Remarquons que le choix d'une autre valeur pour cette constante changera la similitude détectée. En effet, plus la pénalité associée est faible et plus on privilégiera les copies par rapport aux insertions. Cependant, d'un point de vue informationnel, il n'est pas possible de choisir une constante inférieure à la taille du codage nécessaire pour la plus petite position.

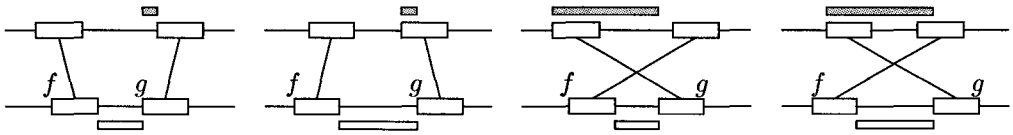
La fonction w_0' fait certes disparaître le défaut mentionné avant mais il n'est toujours pas possible de mesurer correctement l'écart qui existe entre les deux segments d'un même facteur. Le défaut majeur de w_0' est justement que la copie d'un facteur dont les segments sont alignés[11] où d'un facteur dont les segments sont aux extrémités opposées des deux séquences aura le même poids.

Prise en compte des déplacements.

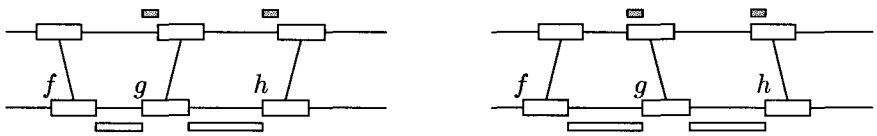
Nous pouvons remédier facilement à cela en considérant la position du facteur g dans T relativement à sa position dans S. Cette position relative est la différence en valeur absolue entre la position de g dans la séquence cible et dans la séquence source. On aboutit ainsi à une fonction de poids plus raisonnable qui favorise plutôt les facteurs tels que leurs segments se trouvent à des positions proches dans les deux séquences.

$$w_1 = 2 \times [f,g]_t + |g| + |p_g^s - p_g^t|$$

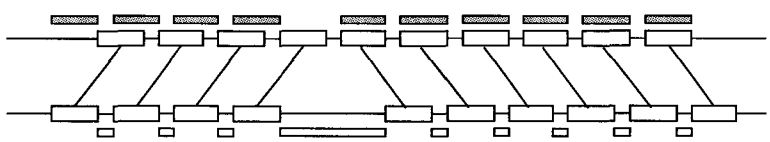
11. Par abus de langage, on dira que deux segments sont alignés si leurs positions respectives dans la source et la cible sont quasiment les mêmes



Observons ce que cela donne avec trois facteurs et le schéma suivant :



Ce schéma révèle une mauvaise propriété de la fonction de coût que nous avons définie. En effet, si il y a copies successives de deux facteurs parallèles (ici g et h), alors le coût de déplacement est compté deux fois. Or, dans un cas où les deux séquences se ressemblent beaucoup, mais où l'une a subi une insertion, il ne paraît pas logique de compter plusieurs fois les déplacements. La figure ci-dessous montre un cas où la fonction w_1 surestime la différence qui peut exister entre les deux séquences :



L'insertion d'un segment est le seul événement évolutif important et lui seul doit être fortement pénalisé. La prise en compte des déplacements pour toutes les copies augmente injustement le poids associé au script.

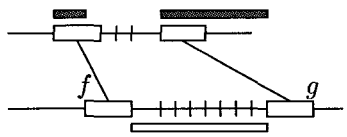
Prenons maintenant le cas de trois facteurs avec un croisement (le déplacement de f n'est pas représenté) :

Pour le schéma de droite, la même remarque que précédemment peut être faite. On compte deux fois un long déplacement alors que les deux segments copiés sont parallèles. L'examen simultané des deux schémas (celui de droite et celui de gauche) révèle un autre problème. Les coûts associés aux déplacements de g et h sont rigoureusement identiques alors que l'interprétation n'est pas du tout la même. En effet, à gauche nous avons la copie de f, puis la copie de g qui croise f, puis la copie de h qui est parallèle à g ; à droite, nous avons la copie de f puis la copie de g qui croise f puis la copie de h qui est "antiparallèle" à g. Il semblerait normal de s'attendre à avoir un poids plus petit pour le script associé au schéma de gauche que pour celui associé au schéma de droite : d'un côté on ne change qu'une fois l'orientation des facteurs alors que de l'autre elle change deux fois.

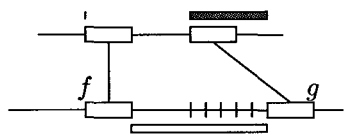
Il se pose également le problème de la différence de longueur des séquences étudiées. En effet, si nous supposons que la séquence source et la séquence cible n'ont pas la même longueur, et

en admettant qu'elles correspondent exactement au séquençage d'une même partie du génome des espèces étudiées, la position relative d'un facteur peut varier suivant la position à laquelle se situe le zéro.

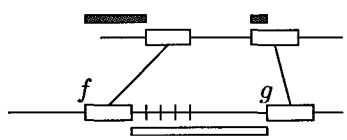
Regardons cela sur un exemple, voici le premier schéma représentant deux séquences et deux facteurs :



Avec ce schéma, nous avons une différence de position de -2 pour le facteur f et de -7 pour g. Considérons maintenant le schéma suivant où les deux premiers facteurs sont "justifiés" à gauche :

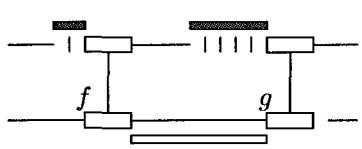


Nous avons ainsi une différence de position de 0 pour f et de -5 pour le facteur g. Par conséquent une représentation avec le second schéma est moins coûteuse qu'avec le premier. Nous pourrions également envisager le schéma suivant :



Il y alors une différence de position de +4 pour f et -1 pour g.

Le problème est qu'il n'est pas satisfaisant de décaler les séquences. Que ce soit pour les problèmes de longueur de séquences ou de multiplication du coût de déplacement de facteurs parallèles, nous ne pouvons choisir de positionner une séquence par rapport à l'autre de manière à optimiser le poids du script. Si les séquences commencent exactement au même locus, alors nous devons conserver cette propriété. Il en est de même pour la fin de la séquence. Si nous représentons les séquences sur une règle, il faut donc que les deux séquences débutent et se finissent aux mêmes positions. De toute façon, il arrivera fréquemment (comme dans le cas du schéma avec de nombreux facteurs parallèles ci-dessus) que les décalages dans un sens ou dans l'autre favorisent ou défavorisent une partie du script. Nous voudrions donc plutôt aboutir à un schéma comme celui ci-dessous (à la manière d'un alignement) :



Les espaces représentent alors les différences de position que nous voudrions réellement prendre en compte. Nous aurons donc une différence de position de -2 pour f et -5 pour g. Cela nous amène à un schéma où nous allons mesurer la différence de position d'un facteur g par rapport au facteur f qui le précède dans le script. Nous considérons alors que le facteur f précédant g a été virtuellement aligné. La différence de positions des segments de g dans la source et la cible seront mesurés par rapport à f. Par exemple, si f se termine en position 10 dans S et 12 dans T et que g débute en position 15 dans S et T alors la différence de position prise en compte pour

g sera de 2 et non de 0. Virtuellement, on considère que f se termine à la même position dans S et T et donc la différence de positions pour les segments de g est donnée par la différence des écarts entre f et g dans S et T.

On comprend bien avec le schéma précédent quelle va être la position prise en compte dans le calcul du poids d'une copie d'un facteur g si il est précédé dans la script par un facteur f ($||x||$ est la valeur absolue de x) :

$$||p_f^s - p_f^t|| - ||p_g^s - p_g^t||$$

Pour le cas où deux facteurs se croisent, cette position devient :

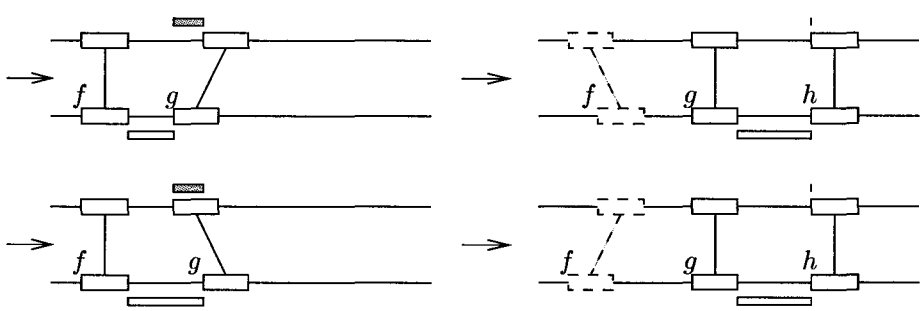
$$||p_f^s - p_f^t|| + ||p_g^s - p_g^t||$$

Croiser deux facteurs coûte donc relativement cher. Ce poids, plus important que celui associé au déplacement de deux facteurs qui ne se croisent pas, assure, dans une certaine mesure, que les croisements repérés par la distance sont potentiellement valables et non dûs au hasard.

La fonction de poids w_2 est donc définie de la manière suivante :

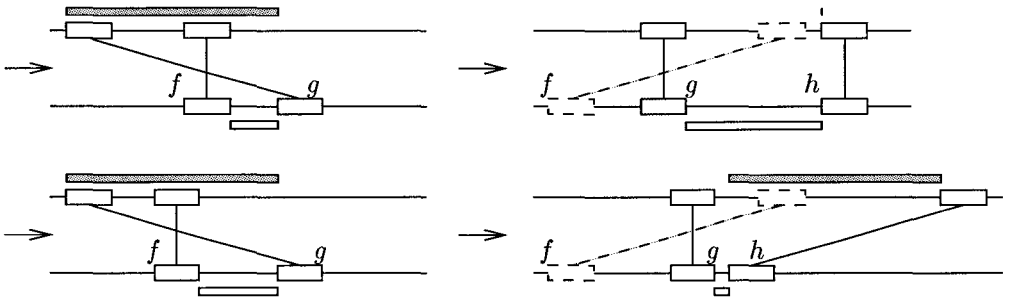
$$w_2(f,g) = \begin{cases} 2 \times [f,g]_t + |g| + \left| ||p_f^s - p_f^t|| - ||p_g^s - p_g^t|| \right| & \text{si } f \text{ et } g \text{ ne se croisent pas,} \\ 2 \times [f,g]_t + |g| + \left| ||p_f^s - p_f^t|| + ||p_g^s - p_g^t|| \right| & \text{si } f \text{ et } g \text{ se croisent.} \end{cases}$$

Reprenons les exemples à trois facteurs. On suppose que le premier facteur, f, a déjà été copié, on observe ensuite ce qui se passe lorsque g est copié (schéma de gauche), puis h (schéma de droite). La longueur des déplacements est représentée par la somme des longueurs des blocs grisés sur les deux schémas. Pour le cas où les facteurs ne se croisent pas, un seul déplacement est compté pour l'ensemble des facteurs parallèles. Cela implique que nous allons privilégier les facteurs ayant peu d'écart, mais dès que l'effort de déplacement aura été fait, alors on supposera que les deux séquences sont maintenant "alignées" sur ce facteur. Il en résulte que les suites de facteurs parallèles sont avantagées.



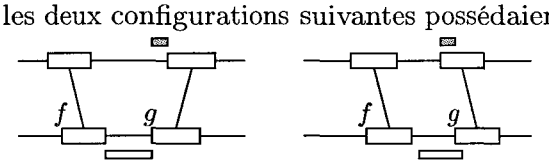
Pour le cas où les facteurs se croisent, on arrive naturellement à faire une distinction entre les facteurs parallèles qui se suivent, ce que w_1 n'était pas capable de faire. Par rapport à w_1, w_2 diminue le coût de déplacement du schéma du haut, profitant du parallélisme des deux facteurs g et h alors qu'elle augmente le coût pour l'autre schéma, pénalisant le caractère antiparallèle de la configuration de g et h. Ce qui nous amène pour le second schéma a avoir deux fois un coût de déplacement important. Cela a pour effet de pénaliser l'intrusion de facteurs parmi une suite

de facteurs parallèles.



Ce codage présente les avantages suivants :

- il conserve la bonne propriété d'un schéma de codage relatif qui favorise les facteurs dont les segments sont bien alignés, sans favoriser ou défavoriser une zone particulière des séquences,
- il favorise des suites de facteurs qui gardent un écart quasi constant. En effet, avec la fonction de codage w_1, les deux configurations suivantes possédaient le même poids :



alors qu'avec w_2, la seconde configuration sera favorisée,
- il pénalise l'introduction d'un croisement mais une fois celui-ci effectué, il favorise les facteurs qui sont parallèles.

Par conséquent la fonction de poids w_2 est celle à retenir parmi les trois envisagées. Elle permet d'identifier des zones similaires de grande densité, c'est-à-dire où l'on va trouver de nombreux facteurs quasi-parallèles et faiblement écartés.

Remarque 6.1 *Avec la fonction de poids w_2, le poids de copie des facteurs virtuels a et z n'est plus nul. En effet, si après copie du dernier facteur les deux segments qu'il reste à insérer n'ont pas la même longueur, il faudra alors considérer un déplacement de la différence des longueurs. Il en est de même pour la copie du premier facteur.*

Remarque 6.2 *La mesure du déplacement est clairement symétrique si on échange les rôles de S et T. Supposons avoir deux facteurs $f = \langle f_s, f_t \rangle$ et $g = \langle g_s, g_t \rangle$. On définit les deux facteurs $f' = \langle f_t, f_s \rangle$ et $g' = \langle g_t, g_s \rangle$. On a immédiatement :*

$$||p_f^s - p_f^t|| - ||p_g^s - p_g^t|| = ||p_{f'}^s - p_{f'}^t|| - ||p_{g'}^s - p_{g'}^t||$$

et

$$||p_f^s - p_f^t|| + ||p_g^s - p_g^t|| = ||p_{f'}^s - p_{f'}^t|| + ||p_{g'}^s - p_{g'}^t||$$

6.2.2 Asymétrie

Nous avons déjà mentionné le caractère asymétrique de la distance de transformation. L'observation des divers schémas présentés ci-dessus montre cette propriété. Deux types de symétrie peuvent être vérifiées :

- la symétrie de lecture : elle sera vérifiée si le sens de lecture de S et T ne modifie pas la distance,

- la symétrie "classique" : elle sera vérifiée si la distance de la source à la cible est la même que la distance de la cible à la source.

Symétrie de lecture

La vérification de cette propriété dépend de la fonction de poids que nous avons choisie. Il est évident qu'avec w_0, l'échange du sens de lecture va modifier la distance obtenue, il en est de même pour w_1 si on n'ajuste pas les débuts de séquences pour que les positions centrales de celles-ci coïncident. Cependant, ce n'est pas le cas avec w_2.

Lemme 6.1 *Avec la fonction de poids w_2, le script $(f_1,f_2,\ldots,f_n)$ est de même poids que le script $(f_n,f_{n-1},\ldots,f_1)$*

Preuve 6.1

D'après la définition de w_2, on a :

$$\begin{aligned} w_2(f_n,\ldots,f_1) &= 2\times[a,f_n]_t + |f_n| + |p_{f_n} - p_a| \\ &\quad + \sum_{i=2}^{n} 2\times[f_i,f_{i-1}]_t + |f_{i-1}| + |p_{f_{i-1}} \pm p_{f_i}| \\ &\quad + 2\times[f_1,z]_t + |z| + |p_z - p_{f_1}| \end{aligned} \tag{6.1}$$

où $p_f = ||p_f^s - p_f^t||$. Posons $j = i-1$, on a alors :

$$\begin{aligned} &\sum_{i=2}^{n} 2\times[f_i,f_{i-1}]_t + |f_{i-1}| + |p_{f_{i-1}} \pm p_{f_i}| \\ &\qquad = \sum_{j=1}^{n-1} 2\times[f_{j+1},f_j]_t + |f_j| + |p_{f_j} \pm p_{f_{j+1}}| \\ &\qquad = |f_1| - |f_n| + \sum_{j=1}^{n-1} 2\times[f_j,f_{j+1}]_t + |f_{j+1}| + |p_{f_{j+1}} \pm p_{f_j}| \end{aligned}$$

L'équation 6.1 s'écrit alors :

$$\begin{aligned} w_2(f_n,\ldots,f_1) &= 2\times[f_1,z]_t + |f_1| + |p_z - p_{f_1}| \\ &\quad + \sum_{j=1}^{n-1} 2\times[f_j,f_{j+1}]_t + |f_{j+1}| + |p_{f_{j+1}} \pm p_{f_j}| \\ &\quad + 2\times[a,f_n]_t + |a| + |p_{f_n} - p_a| \end{aligned} \tag{6.2}$$

Or,

$$\begin{aligned} &2\times[f_1,z]_t + |p_z - p_{f_1}| + 2\times[a,f_n]_t + |p_{f_n} - p_a| = \\ &\qquad 2\times[z,f_1]_t + |p_{f_1} - p_z| + 2\times[a,f_n]_t + |p_a - p_{f_n}| \end{aligned}$$

L'équation 6.2 équivaut à :

$$\begin{aligned} w_2(f_n,\ldots,f_1) &= 2 \times [z,f_1]_t + |f_1| + |p_{f_1} - p_z| \\ &+ \sum_{j=1}^{n-1} 2 \times [f_j,f_{j+1}]_t + |f_{j+1}| + |p_{f_{j+1}} \pm p_{f_j}| \\ &+ 2 \times [f_n,a]_t + |a| + |p_a - p_{f_n}| \end{aligned} \tag{6.3}$$

Comme z et a jouent des rôles symétriques, a devient z et inversement quand on inverse le sens de lecture du script. Par conséquent l'équation 6.3 correspond exactement à l'expression de $w_2(f_1,\ldots,f_n)$. □

Proposition 6.1 *Avec la fonction de poids w_2, la distance de transformation est symétrique par rapport au sens de lecture.*

Preuve 6.1

A partir du lemme précédent, on déduit immédiatement que le script minimal ne change pas suivant le sens de lecture. □

Symétrie

Malheureusement il n'en est pas de même pour la symétrie classique. Nous n'avons pas la propriété que $d(S,T) = d(T,S)$. Cela est bien compréhensible et totalement intrinsèque à la méthode que nous employons : *il n'est pas équivalent de reconstruire S à partir de T et T à partir de S*. En effet, avoir l'une ou l'autre des séquences supposée connue implique que les informations que nous possédons au départ ne sont pas les mêmes.

Une manière simple d'obtenir la symétrie est de considérer la somme des deux distances, et ainsi définir

$$D(S,T) = d(S,T) + d(T,S)$$

On interprète alors cette distance comme la quantité d'information nécessaire pour passer de S à T puis de T à S.

Une propriété des scripts qui les rend intrinsèquement asymétriques est l'autorisation de chevauchements dans la séquence cible et son interdiction dans la séquence source. Dès lors qu'une même partie de la séquence source peut être utilisée, par exemple si l'on copie plusieurs fois un même segement de la séquence source, il n'est plus possible par un script de transformation d'obtenir une distance qui soit symétrique.

Un autre obstacle est la longueur différente des séquences. Nous avons déjà soulevé cette difficulté précédemment lors de la définition de w_2. Celle-ci intervient pour beaucoup dans l'incapacité à concevoir une distance symétrique.

Plaçons nous dans le cas idéal où il n'y a pas de chevauchements mais supposons que les deux séquences S et T sont de longueur différente. Pour une même combinaison de facteurs, il sera alors nécessaire d'avoir au moins une insertion plus grande dans l'un des deux scripts. La distance ne peut alors être symétrique.

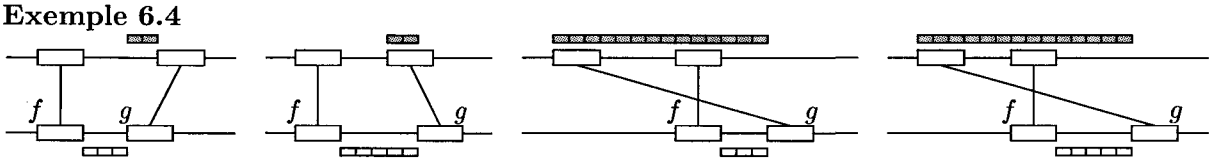
A partir de la définition de w_2 [12] :

$$w_2(f,g) = \begin{cases} 2 \times [f,g]_t + |g| + |p_f - p_g| & \text{si } f \text{ et } g \text{ ne se croisent pas,} \\ 2 \times [f,g]_t + |g| + |p_f + p_g| & \text{si } f \text{ et } g \text{ se croisent.} \end{cases}$$

nous pouvons rapidement cerner le problème. Il apparaît clairement que la mesure du déplacement est symétrique et que les termes $|p_f - p_g|$ et $|p_f + p_g|$ ne varient pas si l'on échange les rôles de S et T. De même, si nous avons des facteurs dont les segments sont de même longueur, ce que nous supposons ici, le terme $|g|$ reste le même. Reste le terme $2 \times [f,g]_t$ qui lui est clairement asymétrique. En effet $[f,g]_s$ n'est pas nécessairement égal à $[f,g]_t$. Un moyen de supprimer cette asymétrie est de considérer le minimum ou le maximum entre ces deux termes.

Nous aboutissons à un codage un peu différent de celui que nous avons adopté jusqu'à maintenant. D'un point de vue compression, il n'est pas nécessaire, et même inutile, de donner des informations qui peuvent être déduites. En l'occurrence, si nous disposons de l'écart dans une séquence entre deux facteurs et du déplacement, nous pouvons en déduire facilement l'écart qu'il y a entre les deux facteurs dans l'autre séquence. Il suffit d'indiquer si nous sommes dans le cas où nous prenons la longueur d'insertion indiquée ou bien dans le cas où nous devons la déduire.

Exemple 6.4



Pour le cas où il n'y a pas de croisement, l'insertion est de longueur 3 pour le schéma de droite et de longueur 5 pour le schéma de gauche. La longueur 3 se déduit de la différence entre la longueur maximale d'insertion et la longueur du déplacement, soit $3 = 5 - 2$. *Pour le cas avec croisement, l'insertion est de longueur 3 à gauche et de longueur 5 à droite. Cette fois 3 se déduit de la longueur du déplacement de laquelle on soustrait la longueur maximale d'insertion et la somme des longueurs des facteurs, soit* $3 = 14 - 5 - (3 + 3)$.

Ecart minimum. Cependant, ici nous n'avons pas à indiquer seulement l'écart mais surtout ce que contient l'écart. En effet, lorsque nous codons une insertion, l'information principale qui doit apparaître est la suite de bases qui composent le segment inséré. Considérer l'écart minimal ne permet pas de décrire les séquences sans perte d'information : il est impossible de coder 5 bases avec seulement 3 bases !

De plus le choix de l'écart minimum modifie fortement la notion de similitude que nous détectons en recherchant le script minimum. Supposons qu'il existe un facteur f entre S et T dont le segment dans S se trouve au début de S et dont le segment dans T se trouve à la fin de T. Lorsque l'on va reconstruire T, ce facteur va être très avantageux. En effet, il va permettre de se trouver très rapidement à la fin de T pour un coût minimum. Le script (f) aura approximativement un coût égal à $2 \times [a,f]_s + 2 \times [f,z]_t$. La reconstruction de S à partir de T aboutit au même résultat.

Ces deux remarques mettent en évidence l'absurdité de considérer l'écart minimum entre $[f,g]_s$ et $[f,g]_t$ pour deux facteurs f et g.

12. $p_f = ||p_f^s - p_f^t||$, soit la différence de position dans la séquence S et dans ma séquence T en valeur absolue.

Ecart maximum. Nous définissons la fonction de poids $w_2^{\max}$:

$$w_2^{\max}(f,g) = \begin{cases} 1 + 2 \times \max([f,g]_s,[f,g]_t) + |g| + |p_f - p_g| & \text{si } f \text{ et } g \text{ ne se croisent pas,} \\ 1 + 2 \times \max([f,g]_s,[f,g]_t) + |g| + |p_f + p_g| & \text{si } f \text{ et } g \text{ se croisent.} \end{cases}$$

Avec cette fonction de poids, et en ne considérant que des scripts dont les facteurs ne se chevauchent pas, nous obtenons effectivement une distance symétrique.

Proposition 6.2 *Si* $(f_1,f_2,\ldots,f_n)$ *est un script sans chevauchement, le poids du script* $(g_1,g_2,\ldots,g_n)$, *avec* $g_i = \langle g_i^s,g_i^t \rangle$ *et* $g_i^s = f_i^t$ *et* $g_i^t = f_i^s$, *calculé avec* $w_2^{\max}$ *est le même que celui de* $(f_1,f_2,\ldots,f_n)$.

Preuve 6.2

La suite de facteurs $(g_1,g_2,\ldots,g_n)$ est effectivement un script puisque qu'il n'existe pas de chevauchement dans S pour le script $(f_1,f_2,\ldots,f_n)$. Ensuite la preuve est évidente. On a immédiatement que l'écart maximum entre f_i et f_{i+1} est le même que l'écart maximum entre g_i et g_{i+1} :

$$max([f_i,f_{i+1}]_s,[f_i,f_{i+1}]_t) = max([g_i,g_{i+1}]_s,[g_i,g_{i+1}]_t)$$

et que la longueur de la représentation de f_i est la même que celle de g_i :

$$|f_i| = |g_i|$$

La symétrie pour le déplacement est évidente (voir remarque 6.2). □

Cette fois nous n'avons plus le même biais qu'introduisait le choix du minimum. Au contraire, cela tend à favoriser plus encore le choix de facteurs bien alignés. D'un point de vue détection de la similarité cette fonction de poids permet de définir une distance satisfaisante.

Pour autant, l'écart maximum n'est pas totalement satisfaisant. D'un point de vue représentation de l'information, il n'est pas la manière la plus économique de représenter l'écart entre deux facteurs. A chaque fois que l'on code une insertion, on code plus d'information que nécessaire, on surévalue ainsi la vraie distance qui sépare S et T.

6.2.3 Réflexivité

Il est très intéressant de noter que la distance de transformation n'est pas réflexive. En effet, le script de poids minimal permettant de construire une séquence S à partir d'elle même sera composé, dans le cas du système générateur $\{ins,cop\}$, de la seule opération '`copie`(S)'. Le coût d'un tel script ne peut pas être nul. Par conséquent, nous choisissons la convention que si $S = T$ alors $d(S,T) = 0$.

6.2.4 Inégalité triangulaire

Puisque nous définissons une "distance", nous devons nous intéresser également à la vérification de l'inégalité triangulaire :

$$\forall\ S,T,U \quad d(S,T) \leq d(S,U) + d(U,T)$$

Contre-exemple. L'inégalité triangulaire n'est pas vérifiée. Il est possible de construire un exemple où celle-ci est mise en défaut. Avec les quatre lettres de l'alphabet des nucléotides et en notant $[x]$ une suite de 45 lettres x, il suffit de prendre $S = [A][T]$, $U = [C][G][G]$ et $T = [A][G]$. On obtient alors :

Distance	Script	Valeur
$d([A][T],[A][G])$	copie($[A]$)	
	insertion($[T]$)	119
$d([A][G],[C][G][G])$	insertion($[C]$)	
	copie($[G]$)	
	copie($[G]$)	158
$d([A][T],[C][G][G])$	insertion($[C][G][G]$)	287

Nous vérifions bien $119 + 158 = 277 < 287$. Plus les suites de lettres serons longues et plus la différence sera grande.

Vérification expérimentale. Nous voyons combien cet exemple est artificiel, tant par la topologie particulière des séquences que par la longueur des segments communs. Il semble donc intéressant de se demander si en pratique l'inégalité triangulaire est souvent vérifiée. Nous utilisons l'échantillon 4 : les 97336 tests ont tous donné un résultat positif à la vérification de l'inégalité. Le même constat a été fait sur l'échantillon 2 (729 tests) et 1 (8000 tests).

6.2.5 Codage des items des fonctions de poids

Une fois qu'une fonction de poids est choisie, et même si elle dispose de bonnes propriétés, le travail de spécification de la distance n'est pas terminée. En effet, la définition de la fonction de poids détermine quels items vont décrire telle ou telle opération. Il reste encore à choisir le **codage** de ces items, c'est-à-dire à choisir la représentation de l'information. Il n'existe pas tout à fait de représentation optimale des items, le codage de chacun peut varier suivant le problème que l'on traite. Par exemple, si il y a de nombreux petits entiers à coder le codage optimal ne sera pas le même que si il y a des grands entiers à coder. Ce codage constitue donc également une part importante dans la définition de la similitude détectée par la distance de transformation.

En respectant le plus possible le principe de représentation de l'information, un script doit être un véritable algorithme de compression. En particulier il doit avoir la propriété d'existence d'un algorithme de décompression (voir chapitre 3). Afin de pouvoir reconstruire la séquence T à partir d'un script et de la séquence S nous devons disposer des informations suivantes (pour la fonction de poids w_2) :

- le type d'opération
- la suite des bases du segment inséré
- la longueur du segment inséré
- la longueur de la copie
- la longueur du déplacement
- le sens dans lequel s'effectue le déplacement

Nous sommes donc amenés à avoir le codage suivant qui fournit une méthode de décompression

($|x|$ représente la longueur binaire de x) :

Opération	Objet codé	Longueur du codage (bits)
insertion	type d'opération	1
	longueur l de l'insertion	$2 \times \|l\|$
	suite de l bases	$2 \times l$
copie	type d'opération	1
	longueur l de la copie	$2 \times \|l\|$
	sens du déplacement	1
	longueur d du déplacement	$2 \times \|d\|$

L'utilisation de $2 \times |x|$ pour représenter un entier est une manière de coder l'entier x si l'on veut pouvoir le retrouver à partir de son codage. C'est une manière d'autodélimiter la représentation d'un entier x.

Avec un tel codage il est effectivement possible de retrouver la séquence T. Si le script dont nous disposons débute par un '1' alors nous savons que c'est une opération de copie. Les bits suivants donnent la longueur de la copie puisque nous avons utilisé un code autodélimité. Le bit suivant indique le sens du déplacement, puis les suivants la longueur de celui-ci. Le bit suivant indique à nouveau le type d'opération. La lecture d'un '0' précise que c'est une insertion. Nous pouvons alors lire la longueur l de l'insertion et disposant de cette information nous savons que les $2 \times l$ bits suivants donnent la suite des bases à insérer. Il existe donc bien une manière non ambiguë de décoder un script décrit avec ce codage.

Des codages dégradés. L'utilité d'utiliser un codage autodélimité peut être discutée ici. En effet l'intérêt d'un tel codage est de disposer d'un algorithme de décompression capable de reconstruire effectivement T à partir de S. Dans notre cas, nous cherchons surtout à évaluer l'effort qu'il y a à faire pour passer de S à T. Une mesure moins précise de la représentation binaire des items de chaque opération est peut être suffisante pour évaluer la ressemblance entre S et T. Cela donne le codage suivant :

Opération	Objet codé	Longueur du codage (bits)
insertion	type d'opération	1
	longueur l de l'insertion	$\|l\|$
	suite de l bases	$2 \times l$
copie	type d'opération	1
	longueur l de la copie	$\|l\|$
	sens du déplacement	1
	longueur d du déplacement	$\|d\|$

Un effet certain de la suppression de l'autodélimitation est que les distances seront plus petites puisque les coûts des opérations d'insertion et de copie seront eux-même plus faibles. Cela implique la modification du point de passage de validité d'une copie par rapport à une insertion. Ainsi, des facteurs plus petits amélioreront d'autant plus le coût des scripts. Ce qui signifie que l'on va plus rapidement préférer une copie à une insertion. La figure 6.7 reprend le même échantillon que celui de la figure 6.1 avec ce codage.

Nous pourrions décider de coder brutalement l'information la plus importante pour chaque opération. Cela revient à ne considérer que trois informations : la longueur de l'insertion, la

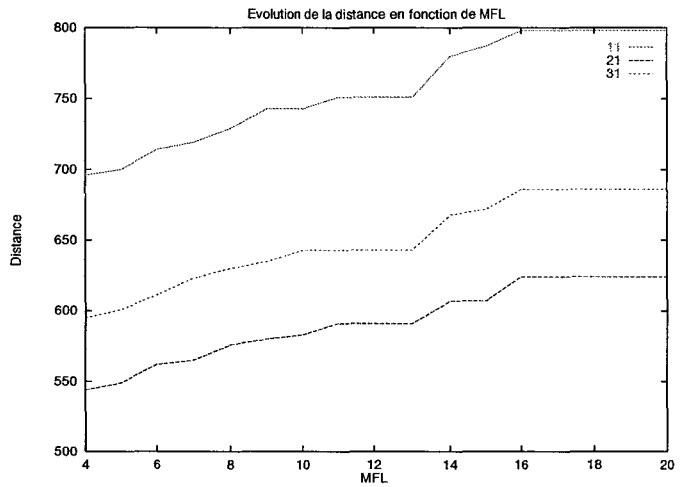


FIG. 6.7 – *Variation de la distance en fonction de MFL pour 3 couples de séquences de l'échantillon 4 (graphe de scripts, codage non autodélimité). La décroissance de la valeur de la distance est plus rapide.*

longueur de la copie, la longueur du déplacement. On aboutit ainsi à ce codage :

Opération	Objet codé	Longueur du codage (bits)
insertion	suite de l bases	$2 \times l$
copie	longueur l de la copie	$\|l\|$
	longueur d du déplacement	$\|d\|$

Cette fois encore, un effet immédiat sera de privilégier très rapidement les copies par rapport aux insertions. Avec un tel codage, la quasi totalité des facteurs communs représentent une opération de copie améliorant le coût du script. L'utilisation du graphe de scripts valides permet alors de disposer d'un très bonne approximation de la distance.

6.2.6 Evaluation des fonctions de poids

Nous utilisons ici l'échantillon 2. Rappelons qu'il est composé de 9 séquence d'ARN ribosomaux d'une longueur variant de 585pb à 876pb.

Influence des fonctions de poids. La figure 6.9 montre la corrélation entre la distance calculée entre S et T avec T pour cible puis avec S pour cible pour les fonctions de poids w_1, w_2 et $w_2^{\max}$. Afin d'obtenir des valeur comparables pour les distances, nous avons utilisé le taux de compression, c'est-à-dire le rapport entre le poids du script minimal permettant d'obtenir T à partir de S et le poids du script ne contenant que l'opération d'insertion de T. Nous notons une asymétrie évidente pour w_1 et w_2 alors que la fonction w_2^{max} vérifie plus largement la symétrie. Nous vérifions également qu'avec cette dernière fonction de poids, la distance est globalement plus élevée. Cela est confirmé avec la figure 6.10 représentant la corrélation entre la distance obtenue avec les fonctions w_1 et w_2, et avec les fonctions w_2 et $w_2^{\max}$. $w_2^{\max}$ donne des distances plus grandes que w_2, ce à quoi nous pouvions effectivement nous attendre.

Il faut étudier qualitativement les scripts obtenus pour vérifier que les les effets que nous avons

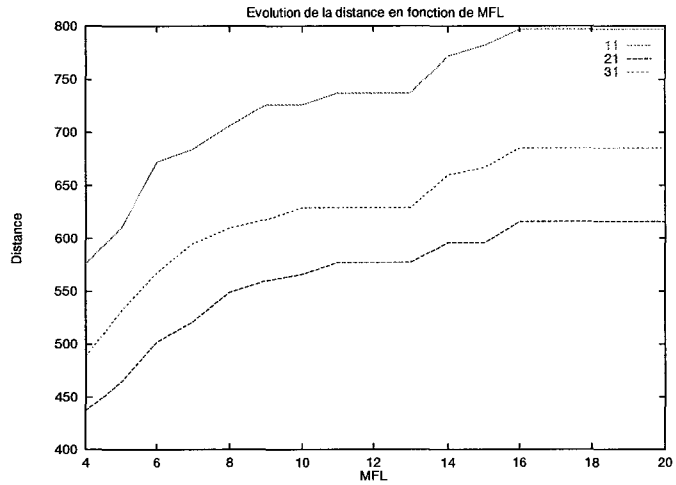


FIG. 6.8 – *Variation de la distance en fonction de MFL pour l'échantillon 4 avec le codage des informations principales. On note une décroissante encore plus rapide que pour le codage non autodélimité.*

associés aux différentes fonctions de poids sont bien constatés. Pour observer que le passage de w_1 à w_2 a effectivement tendance à favoriser les facteurs parallèles, celle-ci étant renforcée lorsque l'on passe à $w_2^{\max}$.

Les figures 6.11, 6.12, 6.13 représentent graphiquement les scripts obtenus pour les fonctions de poids w_1, w_2 et $w_2^{\max}$. Chaque schéma correspond à une comparaison et chaque colonne (respectivement chaque ligne) correspondent à une séquence source (respectivement une séquence cible). La séquence source est représentée en haut alors que la séquence cible se trouve en bas. Les traits horizontaux reliés sont les facteurs du script. Pour se référer facilement à l'un des schémas de chaque figure, on notera i,j le schéma situé en ligne i et en colonne j.

Prenons le schéma 1,1, il apparaît avec w_2 une suite de segments parallèles dans le dernier tiers des séquences. Cette série de segments n'a probablement pas été choisie par w_1 parce que les segments étaient trop courts (en observant le script, on vérifie effectivement que les facteurs ajoutés ont des longueurs variant entre 5 et 12). Deux autres schémas intéressants sont les 1,5 et 2,4. Dans la partie centrale des comparaisons, on note pour w_1 trois facteurs parallèles identifiés puis un facteur vertical qui croise le dernier facteur parallèle. Avec w_2, ce dernier facteur est supprimé et remplacé par un quatrième facteur court parallèle aux trois précédents. Le déplacement du facteur vertical étant plus faible que le déplacement du quatrième facteur parallèle, celui-ci a été choisi alors que le contexte n'était pas favorable. On peut noter un effet du même type pour le schéma 7,1. Pour l'ensemble des schémas, on constate l'ajout de nombreux facteurs parallèles lorsqu'on utilise w_2.

Le passage de w_2 à $w_2^{\max}$ a surtout pour effet d'éliminer des ensembles de facteurs dont l'écart avec les groupes de facteurs leur précédant ou leur succédant est plus grand dans l'une des deux séquences. Ce qui semble tout à fait logique relativement à la définition de $w_2^{\max}$.

Influence des codages des items. La figure 6.14 présente les graphes de corrélation pour la fonction de poids w_2 et les trois codages: autodélimité, non autodélimité et informations principales.

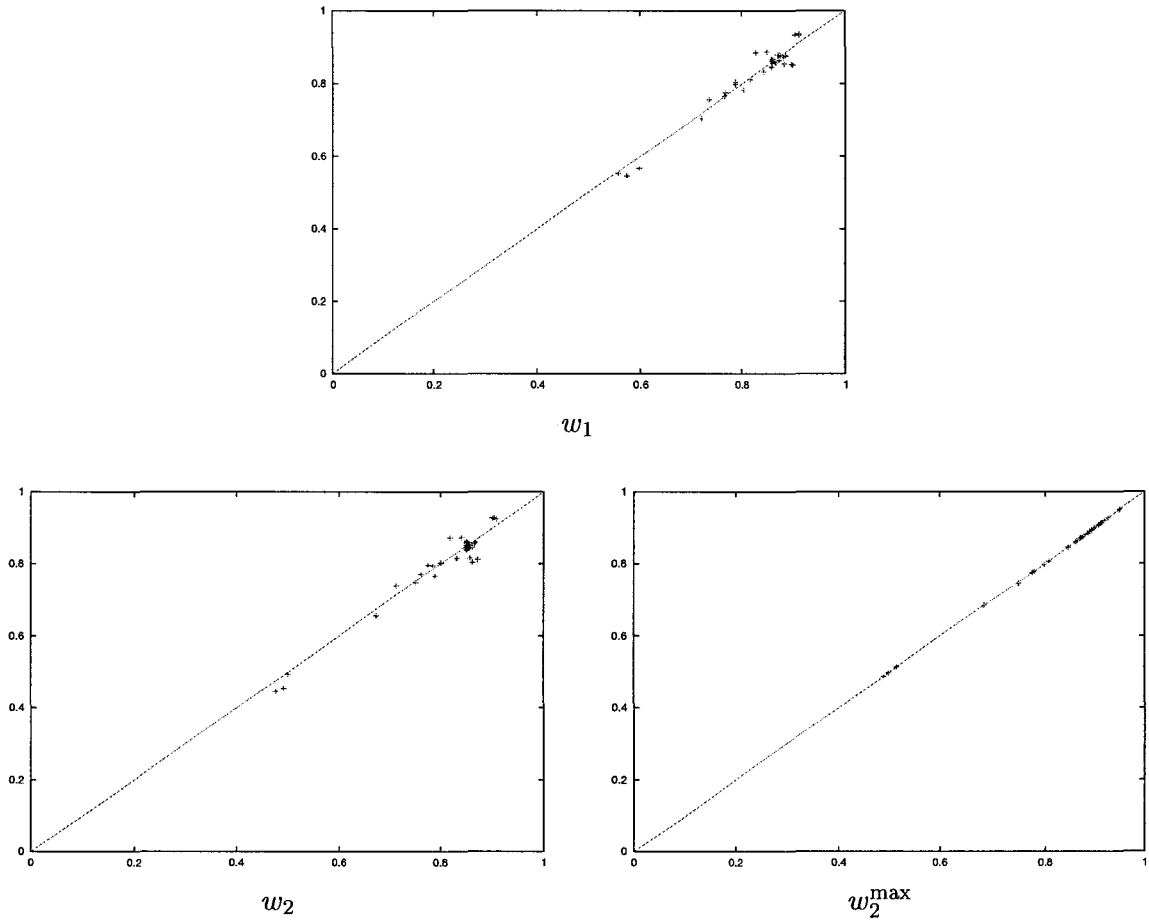


FIG. 6.9 – *Corrélation entre $d(S,T)$ et $d(T,S)$ pour les différentes fonctions de poids. Chaque point représente la valeur de $d(S,T)$ et $d(T,S)$ pour un même couple (S,T). On note une certaine asymétrie pour w_1 et w_2, qui est rétablie avec la fonction de poids $w_2^{\max}$.*

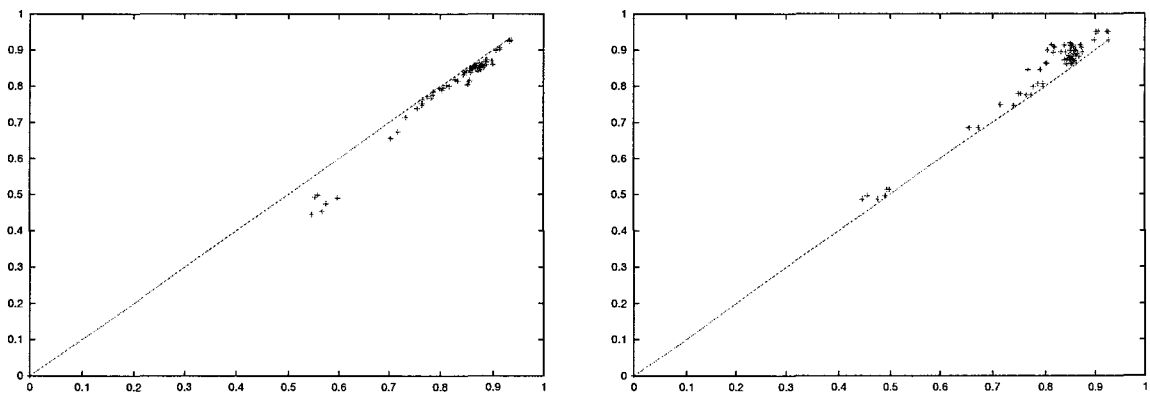


FIG. 6.10 – *Corrélation des fonctions de poids w_1, w_2 et $w_2^{\max}$. w_1 et $w_2^{\max}$ donnent des poids plus élevés que w_2.*

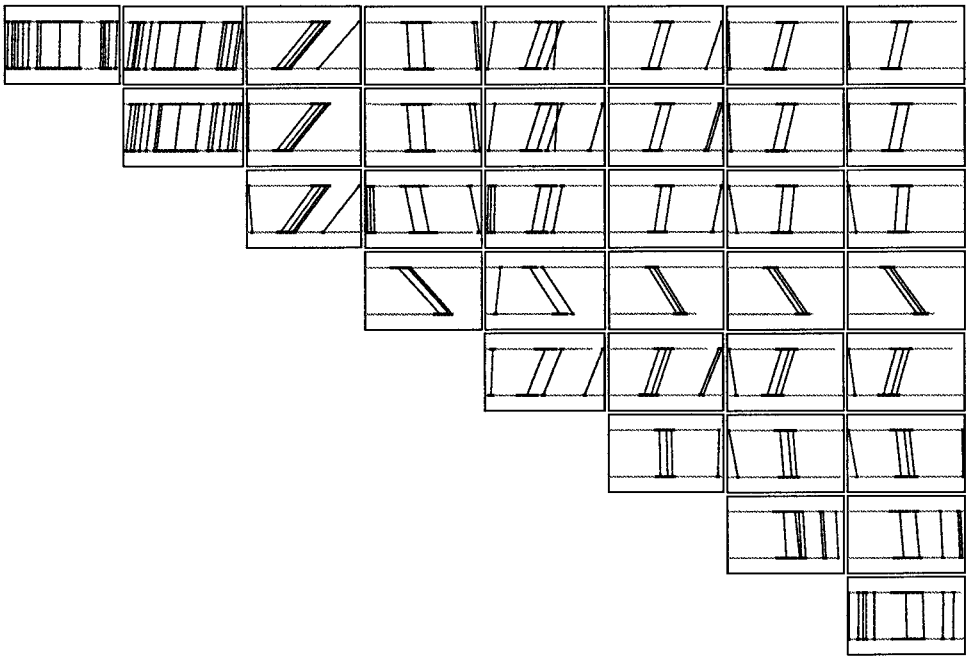


FIG. 6.11 – *Représentation des scripts avec la fonction w_1 et le codage autodélimité pour l'échantillon 2.*

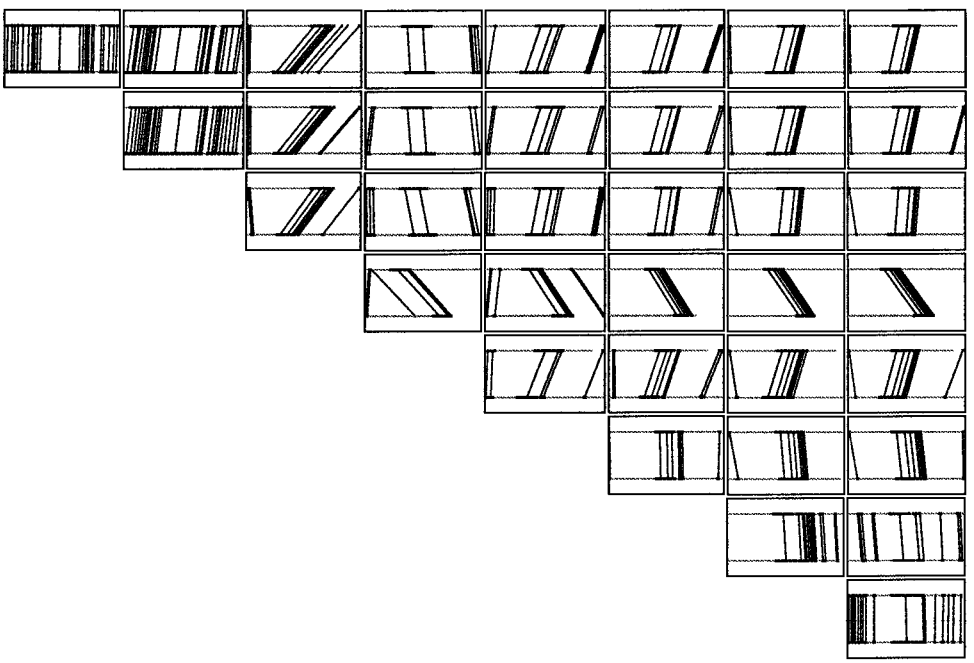


FIG. 6.12 – *Représentation des scripts avec la fonction w_2 et le codage autodélimité pour l'échantillon 2. Par rapport à la fonction w_1, apparaissent des facteurs parallèles qui n'étaient pas sélectionné avant.*

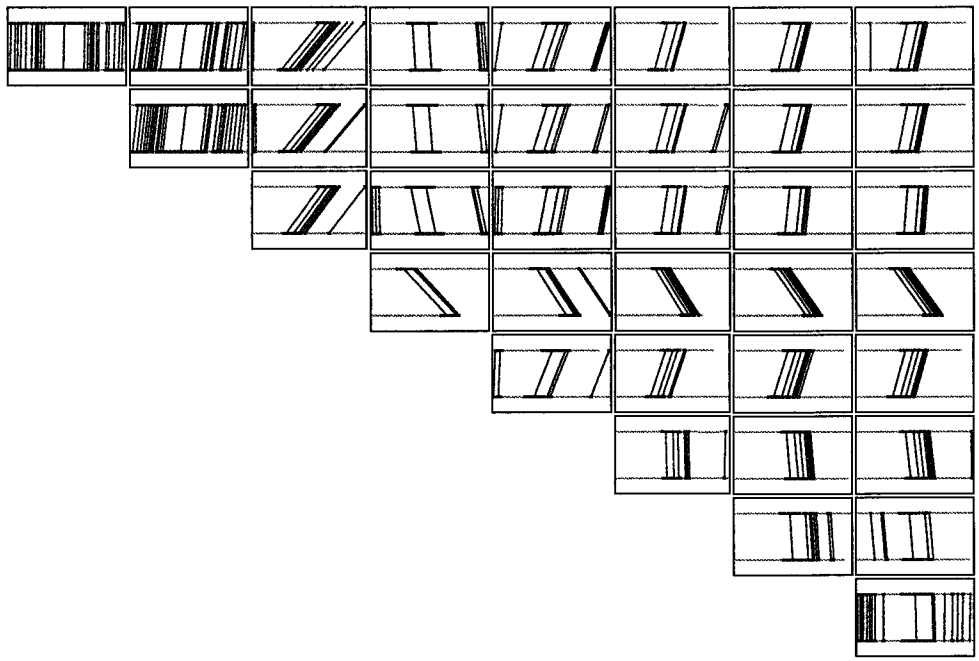


FIG. 6.13 – *Représentation des scripts avec la fonction $w_2^{\max}$ et le codage autodélimité pour l'échantillon 2. Quelques facteurs disparaissent à cause de zones d'insertion trop grandes dans la source.*

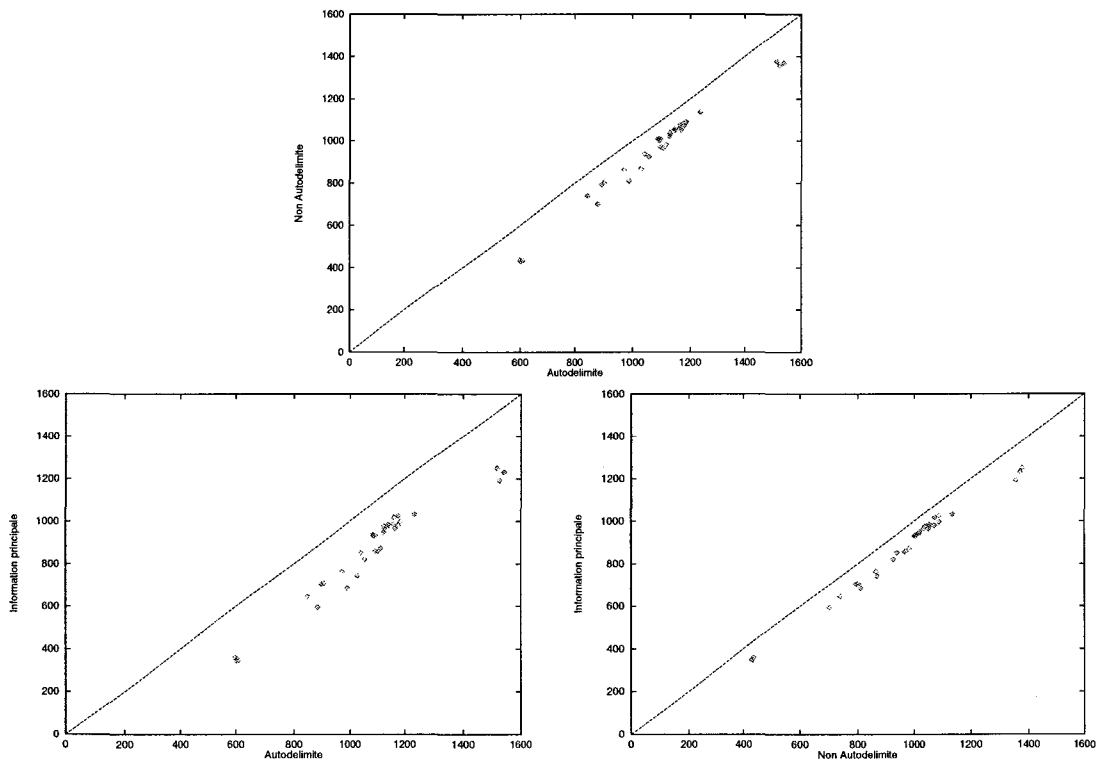


FIG. 6.14 – *Corrélation des codages autodélimité, non autodélimité et information principale.*

Ces graphes montrent que le codage autodélimité aboutit logiquement à des distances plus élevées. Il est intéressant de remarquer que l'ensemble des points reste relativemant homogène et donc que le codage non autodélimité a tendance à diminuer globalement la distance. L'homogénéité est un peu moins bonne avec le codage de l'information principale.

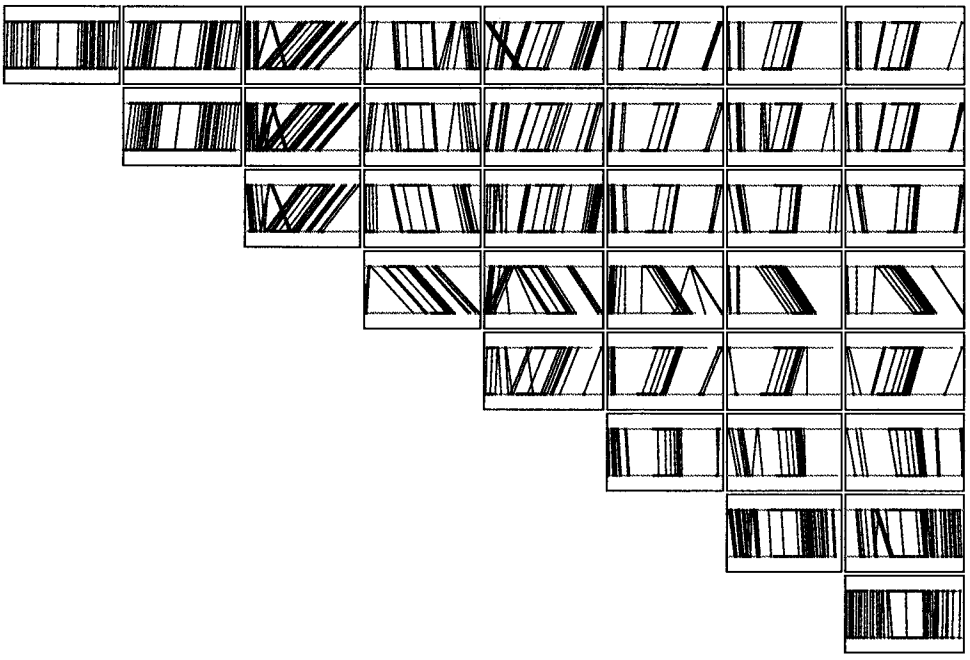


FIG. 6.15 – *Représentation des scripts avec la fonction w_2 et le codage non autodélimité pour l'échantillon 2. L'apparition de croisements doit être validé par l'observation des facteurs mis en jeu.*

L'observation qualitative des schémas (figures 6.12, 6.15 et 6.16) confirme que la dégradation du codage des items diminue le critère de sélection des facteurs copiés.

6.3 Evaluation des graphes de scripts

Dans un premier temps nous revenons sur la vérification des conditions exposées au chapitre précédent pour le graphe de scripts compact et le graphe de scripts valides. Nous proposons ensuite d'évaluer expérimentalement la rapidité de calcul de la distance pour les différents graphes de scripts, puis de mesurer l'erreur obtenue avec le graphe de scripts compact et le graphe de scripts valides.

6.3.1 Vérification des conditions des graphes

Graphe de scripts compact

Le graphe de scripts compact repose sur les trois conditions dont les énoncés sont rappelés ci-dessous. Celles-ci doivent être vérifiées pour que le calcul du meilleur chemin du graphe de

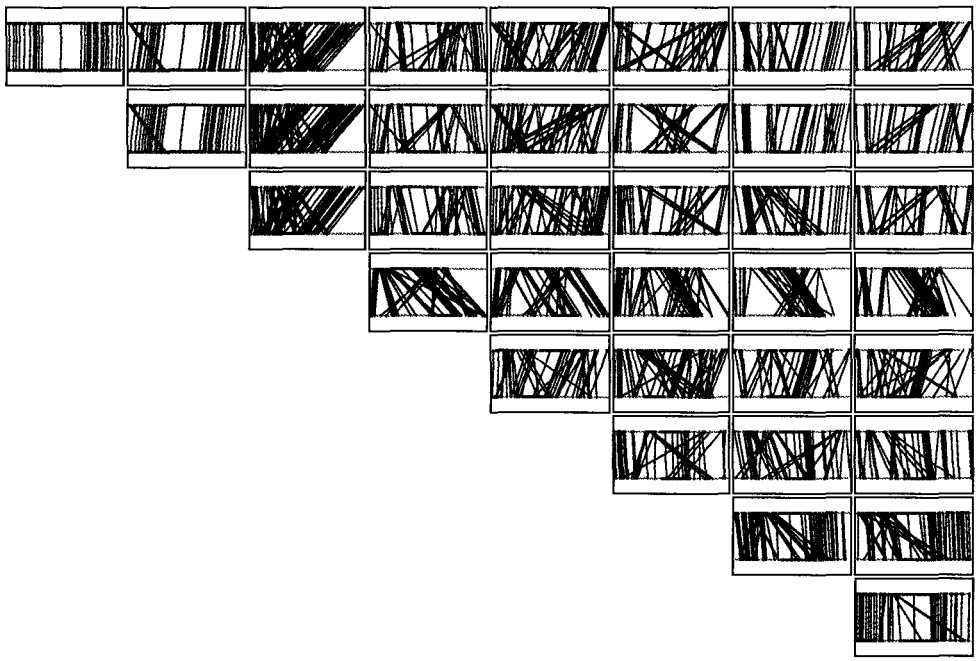


FIG. 6.16 – *Représentation des scripts avec la fonction w_2 et le codage de l'information principale pour l'échantillon 2. Le nombre important de facteurs et de croisements indique que ce codage favorise certainement trop les copies.*

scripts compact corresponde au calcul du meilleur script. Nous nous placons dans le cas où les facteurs sont exacts. Pour chacune d'elles, nous essayons de montrer avec quel degré la condition est vérifiée avec la fonction de poids w_2 et en utilisant le codage autodélimité des items.

Condition 5.1 (des facteurs maximaux) *Pour tous facteurs f,g,h tels que $f <_o g <_o h$, on a :*

$$i(f,g) + c(g) + i(g,h) \leq i(f,g') + c(g') + i(g',h)$$

pour tout sous-facteur $g' \subset g$.

Nous devons donc prouver :

$$2 \times [f,g] + |g| + |d| + 2 \times [g,h] \leq 2 \times [f,g'] + |g'| + |d'| + 2 \times [g',h] \tag{6.4}$$

$|d| = |d'|$ puisque g' est un sous-facteur de g, nous devons donc résoudre

$$2 \times [f,g] + |g| + 2 \times [g,h] \leq 2 \times [f,g'] + |g'| + 2 \times [g',h] \tag{6.5}$$

Il est évident que $2 \times [f,g] < 2 \times [f,g']$, $2 \times [g,h] < 2 \times [g',h]$ et $|g| > |g'|$. Il faut donc regarder dans le détail pour vérifier l'inégalité. Il est assez difficile d'évaluer la véracité de cette inégalité, celle-ci contenant 4 paramètres : l'écart entre f et g, l'écart entre g et h, la longueur de g' et la longueur de g. Nous pouvons réaliser une étude expérimentale en diminuant le nombre de paramètres.

Nous allons supposer que les écarts sont symétriques, c'est-à-dire que $[f,g] = [g,h]$ et $[f,g'] = [g',h]$. L'équation 6.5 se réduit à

$$4 \times [f,g] + |g| \leq 4 \times [f,g'] + |g'| \tag{6.6}$$

Nous avons donc réduit le nombre de paramètres à 3 : l'écart entre f et g, la longueur de g et la longueur de g'. L'étude des courbes traçant les deux membres de l'inégalité 6.6 montre que celle-ci est toujours vérifiée (voir figure 6.17). Cela est encourageant pour la condition des

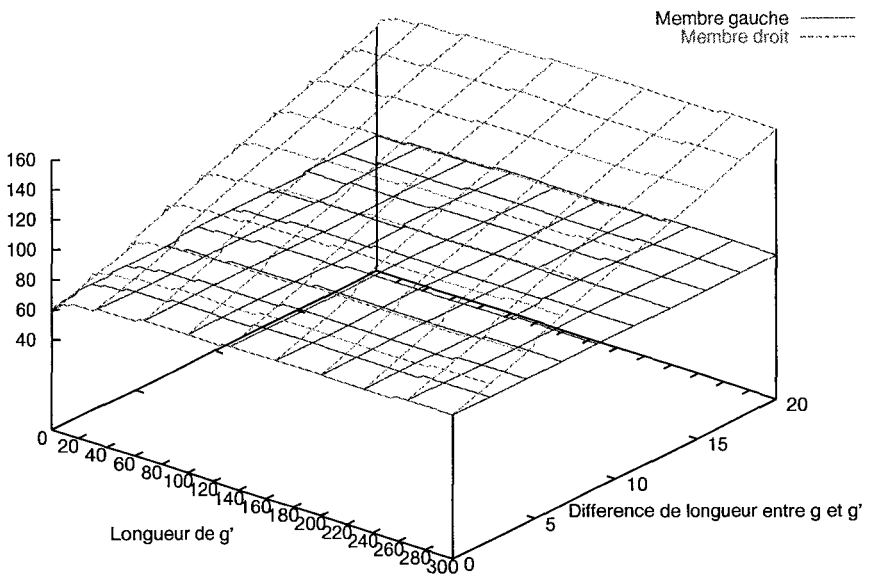


FIG. 6.17 – *Tracé des membres gauche et droit de l'inégalité 6.6 pour un écart entre f et g de 10. Le membre gauche est toujours inféreur au menbre droit, l'inégalité est donc toujours vérifiée.*

facteurs maximaux mais ne montre pas qu'elle est vérifiée pour tous les facteurs. Nous ne pouvons malheureusement pas donner de résultat général sur cette condition.

Condition 5.2 (des arcs ascendants) *Pour tout couple de facteurs f,g tels que* $f <_o g$ *et* $\text{MAX}(g) <_s \text{MAX}(f)$ *on a :*

$$c(h) < c(f) + i(f,g) + c(g)$$

avec $h = \langle f^s, f^t, (g^t + g^l) - f^t \rangle$.

Le déplacement associé à h et f est le même ainsi que celui associé aux successeurs de h et g, il faut donc vérifier :

$$|h| < |f| + 2 \times [f,g] + |g| \tag{6.7}$$

Or, pour tout couple d'entiers positifs a et b, la longueur de la représentation binaire de $(a + b)$ est plus courte ou égale à la somme des longueurs des représentaions de a et b :

$$|a + b| \leq |a| + |b|$$

L'équation 6.7 est donc toujours vérifiée puisque h est tel qu'il débute là où débute f et se termine là où se termine g. Ainsi la longueur de h est la somme des longueurs de f, g, et de l'écart entre f et g.

Condition 5.3 (unicité des arcs descendants) *Pour tout couple de facteurs f',g' tels que $f' <_o g'$ et $\text{MAX}(f') \not<_o \text{MAX}(g')$, avec $f \subset f'$ débutant à la même position que f', se terminant avant f' et $g \subset g'$ se terminant à la même position que g', débutant après le début g' on a :*

$$c(f) + i(f,g) + c(g) \geq c(f') + c(g')$$

Cette fois encore, le déplacement associé à f et f' est le même et le déplacement associé à g et g' également. L'inégalité à vérifier est donc

$$|f| + 2 \times [f,g] + |g| \geq |f'| + |g'| \tag{6.8}$$

En notant a la différence de longueur entre f et f' et b la différence de longueur entre g et g', l'inégalité 6.8 équivaut à

$$|f| + 2 \times [a+b] + |g| \geq |f+a| + |b+g| \tag{6.9}$$

Or on a

$$|f| + |a| + |b| + |g| \geq |f+a| + |b+g|$$

et

$$2 \times [a+b] \geq a + b \geq |a| + |b|$$

On en déduit que la condition d'unicité des arcs descendants est toujours vérifiée.

Remarque 6.3 *En plus de ces conditions, il faut disposer d'une fonction de réduction d'un facteur par rapport à un autre (voir section 5.2.3 page 107). Nous avons implémenté la distance avec une fonction de réduction très simple : si f et g se chevauchent, on conserve g en entier et on prend la partie restante de f. Si le sous-facteur de f ne satisfait pas la contrainte de longueur des facteurs alors la réduction de f par g n'est pas possible.*

Graphe de scripts valides

Condition 5.4 (des facteurs valides) *Pour tout triplet de facteurs f,g,h tels que $f <_o g <_o h$, on a :*

$$i(f,g) + c(g) + i(g,h) < i(f,h)$$

Pour que la condition soit vérifiée, il faut que l'inégalité suivante soit vérifiée :

$$2 \times [f,g] + |g| + |d| + 2 \times [g,h] \leq 2 \times [f,h] \tag{6.10}$$

Si l'on note a l'écart entre f et g, b la longueur de g, c l'écart entre g et h, l'équation précédente s'écrit

$$2 \times [a] + |b| + |d| + 2 \times [c] \leq 2 \times [a+b+c] \tag{6.11}$$

La vérification de cette inégalité est largement dépendante de la valeur de $|d|$. Admettre qu'elle est vraie pour tout triplet de facteurs implique un risque important.

Pour illustrer cela, observons expérimentalement le comportement de l'inégalité en fonction de la longueur du déplacement et ce pour trois facteurs de même longueur.

Longueur des facteurs	Valeur du déplacement où 6.11 est vraie
8	aucune
9	aucune
10	$d \leq 1$
11	$d \leq 7$
12	$d \leq 15$
13	$d \leq 31$
14	$d \leq 63$
15	$d \leq 127$
16	$d \leq 255$

Pour des grands facteurs, la valeur du déplacement peut être assez grande. Par contre pour de petits facteurs (de taille inférieure ou égale à 9), l'inégalité n'est jamais vérifié. Par conséquent la prise en compte systématique de ces petits facteurs introduit une erreur dans le calcul de la distance.

L'utilisation du graphe de scripts valides à partir d'un ensemble de facteurs ne vérifiant pas cette condition, et contenant des facteurs courts, engendrera un taux important d'erreur dans l'estimation de la distance minimale (voir figure 6.26).

6.3.2 Temps d'exécution

Nous utilisons ici les séquences de l'échantillon 3.

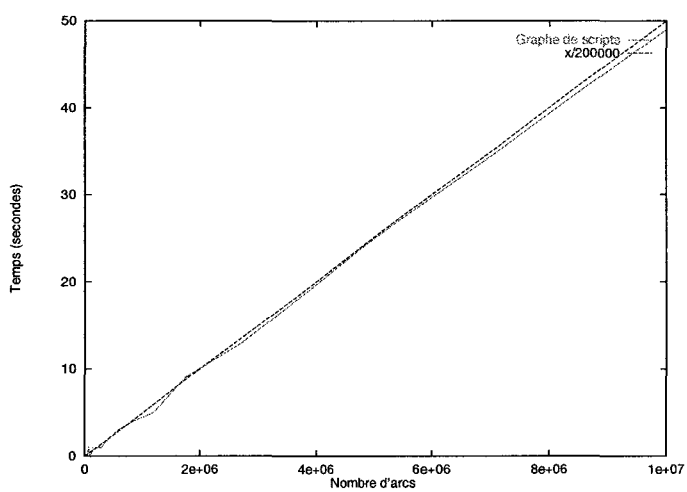


FIG. 6.18 – *Corrélation entre le nombre d'arcs et le temps d'exécution. Evaluation obtenue à partir des valeurs du tableau 6.1.*

La figure 6.18 montre qu'il y a bien corrélation entre le nombre d'arcs contenus dans le graphe et le temps d'exécution. Cela confirme le résultat théorique. La courbe a été obtenue avec

le calcul pour le graphe de scripts. Le coefficient directeur de la droite tracée (1/200000) permet de calculer une approximation du temps de calcul. Rappelons que le nombre d'arcs du graphe de scripts est majoré par $\frac{n^2+3n+2}{2}$ où n est le nombre de facteurs. La figure 6.19 compare la mesure de temps théorique (basée sur le nombre de facteurs attendus) et la mesure de temps réelle. La courbe expérimentale est très proche de la courbe théorique, cela indique que pour cet exemple, le nombre d'arcs effectifs est très proche du nombres d'arcs maximum.

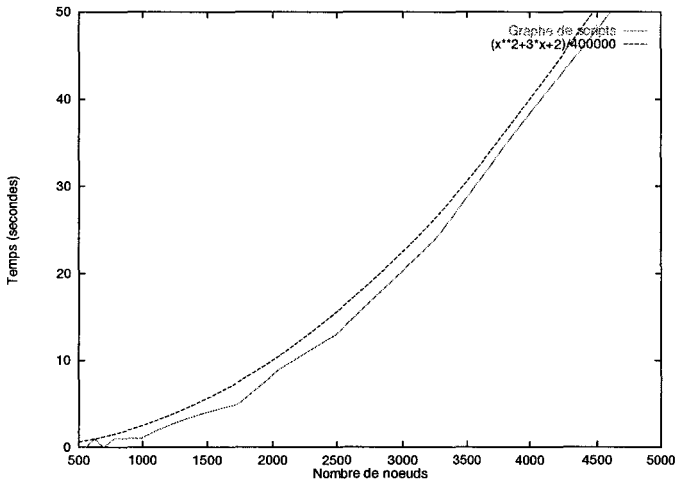


FIG. 6.19 – *Comparaison du temps attendu et du temps obtenu en fonction de nombre de noeuds. Le coefficient estimé de 5e-6 secondes par arc est vérifié.*

D'autres essais ont confirmé la valeur du coefficient multiplicateur permettant de calculer le temps d'exécution en fonction du nombre d'arcs[13]. Nous pouvons donc, avec un calcul simple, estimer le temps maximum d'exécution du calcul de la distance en fonction de la longueur des séquences (on supposera ici que les séquences sont de même longueur l, MFL ayant pour valeur 1, le nombre de facteurs maximum est $\frac{l^3-l}{6}$), la figure 6.20 présente le résultat. Il apparaît clairement que l'estimation théorique maximale que nous proposons et bien plus grande que les résultats obtenus en pratique. Cela est en grande partie dû à la surestimation du nombre de facteurs.

Le tableau 6.1 présente les temps d'exécution du calcul de la distance pour les trois graphes (l'ensemble des facteurs pour le graphe de scripts valides est le même que pour le graphe de scripts). Nous avons fait varier le paramètre de longueur minimale des facteurs afin d'observer le comportement en fonction du nombre de facteurs et du nombre d'arcs du graphe de scripts. La figure 6.21 reprend les même résultats de manière graphique. Nous prédisions dans le chapitre précédent que le gain obtenu avec le graphe de scripts compact était très significatif. Cette prédiction est confirmée ici. Le gain de temps est dû à la très forte diminution du nombre de noeuds, 38% de réduction pour MFL=8 et plus de 95% à partir de MFL=12, et du nombre d'arcs, 81% de réduction pour MFL=8 et plus de 99% de réduction dès MFL=11 (voir tableau 6.1).

Remarque 6.4 *Le nombre d'arcs du graphe de scripts compact correspond au nombre de relâchements. Les prédécesseurs d'un facteur réduit ne sont pas comptabilisés puisque les relâchements sont déjà faits lorsque le facteur est créé. Le rapport temps/nombre d'arcs reste donc valable.*

13. Ce coefficient est bien entendu lié à la machine que nous utilisons. Pour les tests réalisés dans ce chapitre, il s'agissait d'un PC, processeur Pentium II 233Mhz, 32Mo de mémoire, sous Linux.

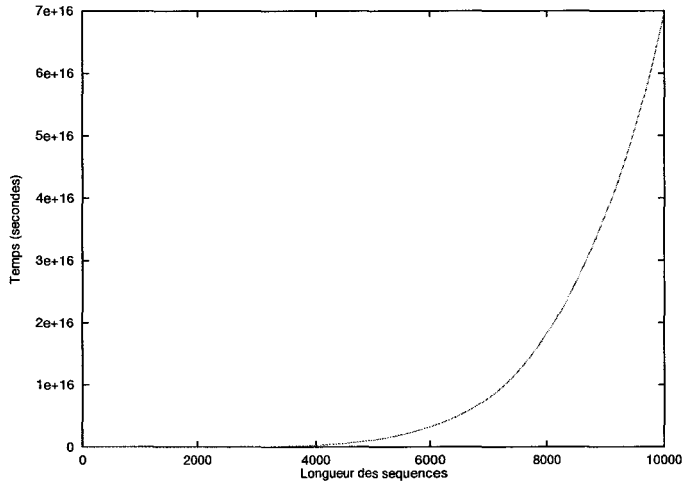


FIG. 6.20 – *Estimation du temps d'exécution en fonction de la longueur des séquences. Résultat basé sur le calcul théorique du nombre de facteurs et sur l'estimation expérimentale du rapport nombre d'arcs / temps d'exécution. Des résultats expérimentaux ont révèlé des temps de calcul bien en deçà de cette estimation. C'est en fait le nombre de sommets, et par conséquent le nombre d'arcs qui sont surestimés.*

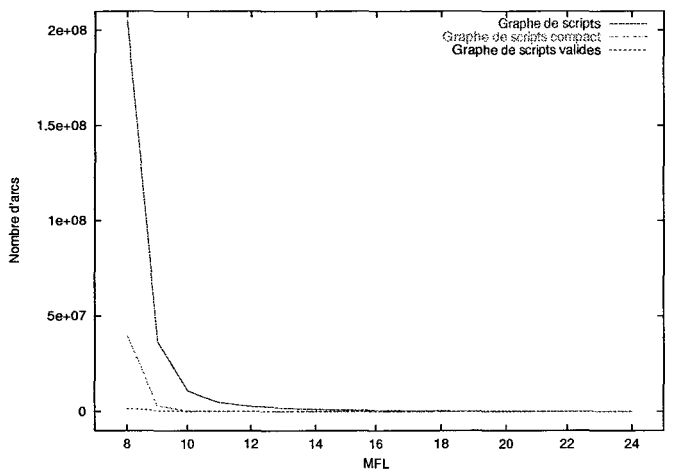


FIG. 6.21 – *Variation du nombre d'arcs en fonction de MFL (échantillon 3). Celui-ci est considérablement réduit lorsque l'on passe du graphe de scripts au graphe de scripts compact. Pour le graphe de scripts valides, le nombre d'arcs est très limité* (1.5e6 *arcs).*

Graphe de scripts

MFL	Nombre de noeuds	Nombre d'arcs	Temps (sec.)	Distance
8	20295	205069875	1146	33771
9	8618	36535513	193	33802
10	4761	10825442	54	33813
11	3250	4831559	24	33843
12	2496	2713229	14	33883
13	2053	1747630	8	33922
14	1737	1186376	6	33933
15	1489	820165	4	33961
16	1287	570357	3	33972
17	1121	398207	2	33990
18	985	280182	2	33999
19	873	199041	1	34011
20	776	139896	0	34011
21	692	97164	1	34016
22	620	67093	0	34048
23	558	46098	0	34048
24	504	31420	1	34092

Graphe de scripts compact

MFL	Nombre de noeuds	Nombre d'arcs	Temps (sec.)	Distance
8	12454	39868266	206	33771
9	2772	3007010	15	33802
10	813	301620	2	33813
11	325	50007	0	33843
12	135	8561	0	33883
13	70	2409	0	33922
14	48	1128	0	33933
15	38	703	0	33961
16	32	496	0	33972
17	26	325	0	33990
18	17	136	0	33999
19	15	105	0	34011
20	14	91	0	34011
21	12	66	0	34016
22	10	45	0	34048
23	10	45	0	34048
24	6	15	0	34092

Graphe de scripts valides

MFL	Nombre de noeuds	Nombre d'arcs	Temps (sec.)	Distance
8	20295	1495441	7	46967
9	8618	490664	3	48659
10	4761	124834	1	43032
11	3250	109055	1	38111
12	2496	97739	0	35467
13	2053	151906	1	34427
14	1737	133754	1	34110
15	1489	118007	1	34034
16	1287	100588	0	34000
17	1121	82982	1	34008
18	985	68526	0	34013
19	873	55235	0	34025
20	776	44153	1	34014
21	692	34630	0	34016
22	620	27056	0	34048
23	558	20443	0	34048
24	504	17489	0	34092

TAB. 6.1 – *Comparaison des temps d'exécution en fonction de la valeur de MFL pour les deux séquences de l'échantillon 3. Les distances obtenues par le graphe de scripts et le graphe de scripts compact sont toutes identiques.*

Remarque 6.5 *Le nombre maximum de noeuds est obtenu avec le graphe de scripts. Il est cependant possible que le graphe de scripts compact contienne plus de noeuds que le graphe de scripts. Cela provient du fait que nous n'autorisons pas un facteur réduit à être utilisé dans un autre chemin que celui pour lequel il a été calculé, il se peut donc qu'un même facteur réduit soit créé plusieurs fois. Nous n'avons observé ce phénomène que très rarement et pour des valeurs de MFL très petites (plus petites que la valeur par défaut).*

6.3.3 Erreur sur les résultats

Nous utilisons principalement dans cette section les séquences issues de l'échantillon 4. Notons que le calcul des 2070 comparaisons a nécessité environ 1 heure de calculs avec le graphe de scripts, 4 minutes pour le graphe de scripts compact et a duré 22 minutes pour le graphe de scripts valides (les temps donnés ici comprennent également le temps de calcul de l'ensemble des facteurs).

Pour estimer l'erreur qui existe lorsqu'on utilise le graphe de scripts compact et le graphe de scripts valides nous avons tracé deux types de courbes. L'une représente la corrélation entre les distances (figures 6.22 et 6.24), nous mesurons ainsi l'écart entre les valeurs. L'autre représente le rapport entre les distances (figures 6.23 et 6.25) ce qui nous permet d'apprécier le degré d'erreur.

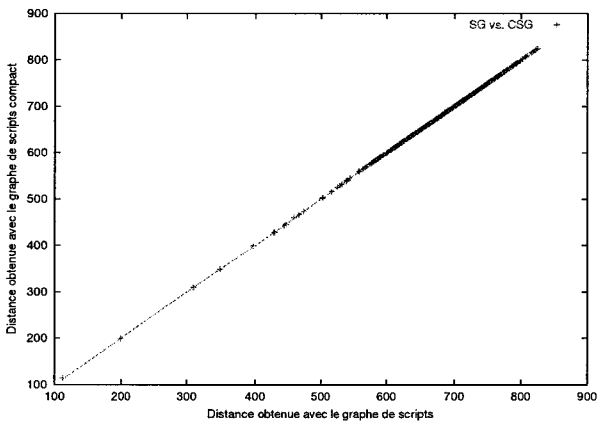


FIG. 6.22 – *Corrélation entre le calcul de la distance avec le graphe de scripts et avec le graphe de scripts compact. Les distances obtenues par les deux méthodes semblent parfaitement corrélées.*

Les distances calculées avec le graphe de scripts compact sont souvent identiques à celles calculées exactement. De plus, lorsque la distance est différente, l'erreur est assez faible. On note 2% d'erreur au maximum et 2,7% de cas avec erreur (28 erreurs sur 1035 calculs). L'erreur maximale est obtenue pour un couple de séquences très proches. Le choix de coupe brutale de notre fonction de facteurs réduits peut expliquer l'erreur de ce cas (la distance avec la graphe de scripts est 112 alors qu'elle est de 114 avec le graphe de scripts compact).

Pour le graphe de scripts valides, l'erreur est plus importante. Cela semble tout à fait logique et en accord avec l'étude expérimentale que nous avons faite plus haut. On remarque que plus la distance est grande et plus l'erreur est importante. Cela est logique : plus la distance est grande et moins de facteurs ont été sélectionnés par la graphe de scripts parce qu'ils n'apportaient aucune information ; le graphe de scripts valides ne fait pas de discrimination sur les facteurs. Notons également que là ou l'erreur atteignait 2% pour la graphe de scripts compact, elle atteint ici 42%

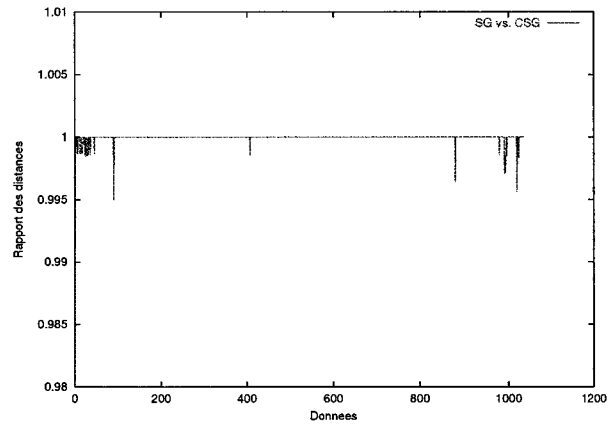


FIG. 6.23 – *Rapport entre la distance obtenue par le graphe de scripts et le graphe de scripts compact pour tous les couples. L'erreur maximale est de 2%.*

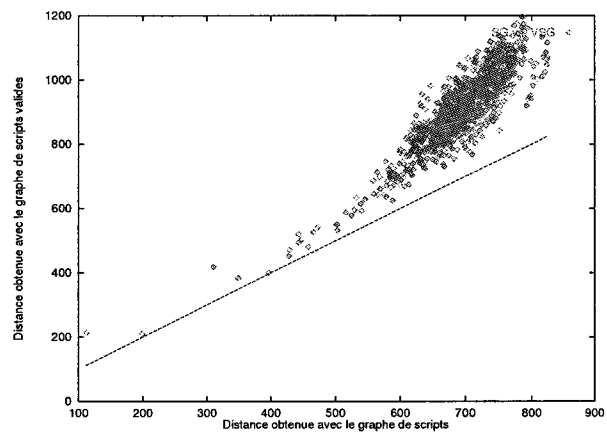


FIG. 6.24 – *Corrélation entre le calcul de la distance avec le graphe de scripts et avec le graphe de scripts valides. L'évaluation est globalement plus grande avec le graphe de scripts valides.*

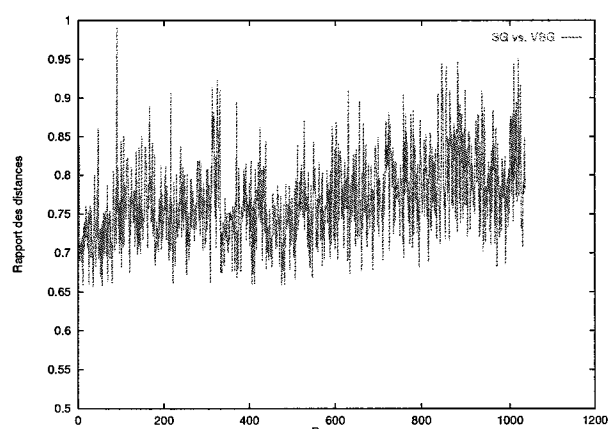


FIG. 6.25 – *Rapport entre la distance obtenue par le graphe de scripts et le graphe de scripts valides pour tous les couples. Le taux d'erreur est généralement grand et peut atteindre 35%.*

au maximum et 5% au minimum, la moyenne se situant autour de 20 à 25%.

Nous voulons maintenant étudier le taux d'erreur en fonction de la valeur du paramètre MFL. Nous reprenons les deux séquences de l'échantillon 3, dont les résultats d'exécution sont donnés tableau 6.1. La figure 6.26 trace la variation de la distance. Pour le graphe de scripts et le graphe de scripts compact la distance calculée ne varie pas (nous avions déjà vérifié cela juste avant). Avec le graphe de scripts valides, la courbe de la distance augmente avec l'ajout de petits facteurs. Avec cette résolution, tous les facteurs disponibles sont inclus dans le script, par conséquent il n'y a plus de discrimination et les facteurs ne sont pas éliminés. Le graphe de scripts valides n'est à utiliser que lorsqu'on dispose d'un ensemble de facteurs que l'on connaît bien ou lorsque le paramètre MFL est choisi suffisamment grand pour ne conserver que des facteurs qui sont très certainement informatifs.

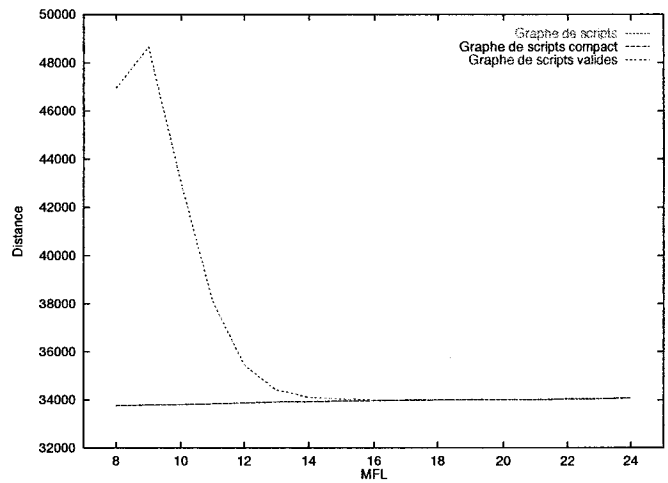


FIG. 6.26 – *Comportement de la distance en fonction du paramètre MFL pour les trois graphes de scripts. On note un accroissement de la distance pour le graphe de scripts valides lorsque MFL diminue, cela provient du caractère glouton de l'algorithme qui inclut dans le script le maximum de facteurs possibles.*

6.4 Résumé

Nous disposons maintenant d'une distance dont nous connaissons un peu mieux les propriétés. L'étude des fonctions de poids et des codages permet de mieux comprendre les résultats et la façon dont s'opérent les choix lors de la recherche du script minimum. Les résultats expérimentaux sur l'inégalité triangulaire sont encourageants, le défaut majeur de la distance étant son asymétrie.

Nous avons également validé l'algorithme basé sur le graphe de scripts compact, qui bien qu'heuristique, donne une très bonne approximation de la distance. Rappelons que sur les échantillons que nous avons traités, nous n'avons noté que quelques erreurs (moins de 1% des comparaisons), et que l'erreur était faible (moins de 2%).

Enfin, nous pouvons utiliser une borne par défaut pour MFL, qui trouve une justification théorique. Nous pensons que cette valeur théorique est corrélée avec le calcul expérimental du point d'effort, celui-ci étant supérieur ou égal à la valeur théorique par défaut.

Chapitre 7

Analyse phylogénétique des isopodes terrestres

Ce travail s'est fait en collaboration avec Didier Bouchon et Alice Michel-Salzat du Laboratoire de Génétique et Biologie des Populations de Crustacés, UMR CNRS 6556, de l'Université de Poitiers.

La bactérie *Wolbachia* infecte une grande partie des arthropodes [BRJ98]. Cette bactérie a des effets sur la reproduction des organismes qu'elle infecte. Elle est en particulier responsable de la féminisation chez les isopodes, seul ordre à être affecté par *Wolbachia* : les mâles infectés par cette bactérie deviennent des femelles fonctionnelles. La quasi-totalité des espèces d'isopodes terrestres est infectée et c'est dans le sous-ordre des *Oniscidea* (*Crustacea, Isopoda*) que le nombre de cas infectés est le plus grand. Des discordances entre la phylogénie des *Wolbachia* et celle de ses hôtes révèlent l'existence de transferts horizontaux entre espèces.

Les relations phylogénétiques entre les *Oniscidea* sont très controversées bien que de nombreuses études aient été réalisées [Van43, Van65, Erh98]. Les phylogénies reconstruites jusqu'à maintenant sont basées sur des critères morphologiques. Pour affiner la comparaison entre la phylogénie des *Wolbachia* et celle des *Oniscidea*, Alice Michel-Salzat et Didier Bouchon ont entrepris de reconstruire une phylogénie moléculaire des isopodes afin d'éclaircir quelques points : la position des *Oniscidea* au sein des isopodes, les relations entre deux familles d'isopodes terrestres *Tylidae* et *Ligiidae*, et les relations entre les familles au sein de la section des *Crinocheta*.

L'utilisation de différentes méthodes à partir des données moléculaires a permis de répondre à certaines questions mais en laisse d'autres en suspens. L'utilisation d'une méthode supplémentaire peut valider certains résultats ou affiner la compréhension des problèmes posés. La distance de transformation est une nouvelle méthode susceptible de donner des réponses complémentaires. L'approche de la mesure de la similarité que nous proposons est totalement différente et on ne s'attend pas a priori à ce qu'elle donne des résultats identiques à ceux des méthodes disponibles. L'obtention de relations phylogénétiques similaires est une validation des propositions actuelles, et aussi une validation de notre méthode. L'observation de relations différentes demande une étude plus approfondie de la pertinence de l'évaluation de la similarité par notre approche, relativement aux méthodes classiques. Nous proposons donc de reconstruire une phylogénie en utilisant la distance de transformation et d'analyser celle-ci par rapport aux problèmes généraux posés ci-dessus.

Nous débutons ce chapitre par une introduction de la problématique et et un état de l'art des relations phylogénétiques des isopodes. Nous présentons ensuite les données que nous traitons ainsi que les méthodes que nous appliquons pour l'analyse de celles-ci, en particulier nous décrivons les paramètres choisis pour la distance de transformation. Nous décrivons ensuite les résultats puis commentons ceux-ci dans la dernière partie.

7.1 Introduction

Les *Oniscidea* sont un sous-ordre des isopodes composé quasi-exclusivement d'espèces terrestres. Une première classification de ceux-ci a été proposée par Vandel en 1943 [Van43], basée sur la morphologie de l'appareil génital masculin. Il proposa une première hypothèse phylogénétique considérant une polyphylie des *Oniscidea* en deux familles dérivant indépendamment à partir des isopodes aquatiques. Le même auteur proposa en 1965 [Van65] d'ajouter une troisième famille en se basant sur des données morphologiques et géographiques. Il suppose donc que les *Oniscidea* sont séparés en trois familles : les *Tylomorpha*, groupe frère du sous-ordre *Valvifera* ; les *Diplocheta* et *Crinocheta*, groupes frères du sous-ordre des *Flabellifera* ; et les *Trichoniscoidea* dont l'origine n'est pas déterminée.

Des études cladistiques récentes [Erh98] défendent l'hypothèse opposée : que les *Oniscidea* forment un groupe monophylétique, ayant divergé une seule fois des isopodes aquatiques. Mais la position des *Oniscidea* au sein des isopodes est incertaine : certaines études concluent à une dérivation à partir d'espèces du sous-ordre *Asellota*, d'autres les placent comme groupe frère des *Valvifera*. Les positions des familles à l'intérieur de ce groupe ne sont pas claires non plus. En particulier, la position des familles *Tylidae* et *Ligiidae* pose problème. Il n'existe pas aujourd'hui de consensus sur une phylogénie des isopodes terrestres. Une hypothèse pour expliquer cette difficulté est que les convergences morphologiques sont tellement nombreuses qu'elles masquent le signal phylogénétique.

Michel-Salzat et Bouchon [MSB00] ont proposé une première étude phylogénétique des isopodes basée sur des données moléculaires. Ils ont séquencé le gène codant l'ARNr mitochondrial 16S chez 44 crustacés terrestres et aquatiques. Les détails concernant ces données figurent section 7.2.1. L'étude phylogénétique a porté sur ces 44 séquences auxquelles ont été ajoutées deux branchiopodes (*Artemia salina* et *Artemia franciscana*). L'analyse des données a révélé que deux espèces étaient particulièrement saturées en transitions (*Helleria brevicornis* et *Cyathura carinata*). Cette saturation pouvant conduire à des positions artéfactuelles dans les arbres, elles ont été retirées de l'analyse. Plusieurs reconstructions ont été réalisées, aussi bien par des méthodes de distance (modèle General Time Reversible [ROMM90], voir section 2.2.2, et méthode Neighbor-Joining [SN87], voir section 7.2.3) que par maximum de vraisemblance. Une première reconstruction sur l'échantillon complet a montré que deux espèces étaient sur des branches longues (*Actaecia bipleura* et *Apseudes latreillei*). Le phénomène d'attraction des longues branches fait que celles-ci ont tendance à attirer les autres espèces vers elles. Ces deux espèces furent également retirées de l'échantillon. Un arbre de consensus a été produit à partir de 1000 réplicats pour un échantillon de 42 espèces (figure 7.4). L'analyse de cet arbre, qui répond partiellement aux problèmes posés, aboutit aux conclusions suivantes. La monophylie des *Oniscidea* est confirmée et les sections des *Crinocheta* et *Synocheta* sont monophylétiques et groupes frères. Les *Ligiidae* et les *Tylidae* sont à la base par rapport aux autres familles des *Oniscidea*. Les positions relatives des isopodes terrestres et des isopodes aquatiques ne sont pas résolues.

7.2 Matériel et protocole expérimental

7.2.1 Les séquences et leur alignement

L'échantillon dont nous disposons est constitué de 46 crustacés terrestres et aquatiques (la classification est donnée dans le tableau 7.1). Parmi ces espèces, 38 sont des isopodes terrestres et aquatiques appartenant à 5 sous-ordres (*Anthuridea, Flabellifera, Asellota, Valvifera* et *Oniscidea*). On retrouve en groupe externe 8 séquences : 2 espèces de *Branchiopoda* (*Artemia franciscana* et *Artemia salina*), 4 espèces appartenant à 3 ordres de *Péracarides* (*Amphipoda, Cumacea* et *Tanaidacea*) et 2 espèces appartenant à un ordre des *Eucarida* (*Decapoda*). Parmi les *Isopoda* le sous-ordre des *Oniscidea* est constitué quasiment exclusivement d'espèces terrestres.

Les séquences sur lesquelles est basée l'analyse sont le résultat du séquençage partiel du gène codant pour la grande sous-unité de l'ARN ribosomique mitochondrial (ARNr 16s mt). Les séquences ont été spécifiquement extraites par Michel-Salzat pour la reconstruction de la phylogénie discutée dans [MSB00]. Ce gène, connu pour évoluer lentement de manière générale, est un bon marqueur phylogénétique. Ces ARN ribosomiques ont la particularité d'être uniquement transcrits et ne sont pas traduits en protéines. Les contraintes de la structure secondaire et tertiaire de l'ARN font qu'il est possible de trouver dans ces séquences des segments similaires qui ne sont pas nécessairement alignés.

La longueur des séquences varie de 319 bases à 415 bases. Les séquences sont disponibles dans les banques de données sous les numéros d'accès AJ388070 à AJ388113. L'alignement a été fait pour les 36 séquences d'isopodes à l'aide de ClustalW [THG94]. Ont été ensuite ajoutés les groupes externes. L'alignement obtenu a été retravaillé à la main en tenant compte des contraintes des structures secondaires sur la base des structures connues de *Drosophila melanogaster*, *Drosophila yakuba*, *Penaeus notialis*, *Artemia franciscana* et *Artemia salina*. L'alignement final est donné dans l'annexe A.

On note une large délétion entre les positions 160 et 210 variant de 30 bases à 54 bases, spécifique aux *Crinocheta*. Une insertion de 3 bases en position 361 est spécifique à la famille des *Porcellionidae*.

Parmi ces séquences, quatre posent problème lors des reconstructions classiques. Les deux séquences *Helleria brevicornis* et *Cyathura carinata* présentent une forte saturation en transitions (figure 7.1). La saturation est mesurée grâce à un diagramme sur lequel on fait apparaître le taux de transitions contre le taux de transversions pour chaque couple de séquences alignées. Cela donne un nuage de point qui indique s'il y a ou non saturation. Si le nuage a une forme proche d'une droite, c'est qu'il n'y a pas saturation. Si au contraire on observe un plateau, c'est qu'il y a saturation. Cette saturation fait que la position de ces séquences dans une reconstruction phylogénétique doit être considérée avec méfiance. Il vaut donc mieux ne pas les considérer.

Deux autres séquences *Apseudes latreillei* et *Actaecia bipleura* ont des branches longues. L'inconvénient de ces longues branches est que celles-ci ont tendance à attirer les autres espèces. Ce phénomène d'attraction des longues branches a été décrit [Fel78].

7.2.2 Les paramètres de la distance de transformation

Au contraire des méthodes classiques, le calcul des distances pour tous les couples d'espèces ne se fait pas à partir d'un alignement multiple. La distance de transformation ne nécessite

Ordre *Sous-ordre*	Infra-Ordre *Section*	Super-Famille *Famille*	Espèce		Lg.	Numéro d'accès
Branchiopoda			*Artemia franciscana*		394	X69067
			Artemia salina		393	X12965
Decapoda						
Dendrobranchiata		Penaeoidea				
		Penaeidae	*Penaeus japonicus*		413	AJ388112
			Penaeus monodon		415	AJ388113
Cumacea		Bodotriidae	*Cumopsis fagei*		371	AJ388111
Amphipoda						
Gammaridea		*Gammaridae*	*Gammarus locusta*		363	AJ388070
			Gammarus zaddachi		360	AJ388071
Tanaidacea						
Apseudomorpha		Apseudoidea				
		Apseudilae	*Apseudes latreillei*	A	366	AJ388110
Isopoda						
Anthuderia		*Anthuridae*	*Cyathura carinata*	A	328	AJ388072
Flabellifera		*Cirolanidae*	*Eurydice affinis*	A	384	AJ388073
		Sphaeromatidae	*Sphaeroma serratum*	A	384	AJ388074
			Sphaeroma tessieiri	A	379	AJ388075
			Campecopea hirsuta	A	391	AJ388076
Asellota		Aselloidea				
		Asellidae	*Asellus aquaticus*	A	378	AJ388077
		Janiroidea				
		Janiridae	*Jaera albifrons*	A	384	AJ388078
			Janira sp.	A	368	AJ388079
Valvifera		*Idoteidae*	*Idotea chelipes*	A	387	AJ388080
			Idotea schmittii	A	386	AJ388081
			Idotea woesneseinskii	A	387	AJ388082
Oniscidea	Tylomorpha	*Tylidae*	*Tylos europaeus*	T	386	AJ388083
			Helleria brevicornis	T	404	AJ388084
	Ligiamorpha					
	Diplocheta	*Ligiidae*	*Ligia oceanica*	T	388	AJ388085
			Ligidum hypnorum	T	375	AJ388086
	Synocheta	Trichoniscoidea				
		Trichoniscidae	*Oritoniscus flavus*	T	383	AJ388087
			Trichoniscus pusillus	T	372	AJ388088
			Haplaphtalmus mengei	T	377	AJ388089
	Crinocheta	Oniscoidea				
		Oniscidae	*Oniscus asellus*	T	341	AJ388090
			Chaetophiloscia elongata	T	348	AJ388091
		Platyarthridae	*Platyarthrus hoffmanseggi*		347	AJ388092
		Armadilloidea				
		Actaeciidae	*Actaecia bipleura*	T	375	AJ388093
		Armadillidae	*Armadillo officinalis*	T	345	AJ388094
			Cubaris murina	T	335	AJ388095
		Armadillidiidae	*Armadillidium vulgare 2*	T	337	AJ388096
			Armadillidium vulgare 1	T	335	AJ388097
			Armadillidium nasatum	T	338	AJ388098
			Armadillidium album	T	339	AJ388099
			Eluma purpurascens	T	340	AJ388100
		Cylisticidae	*Cylisticus convexus*	T	355	AJ388101
		Porcellionidae	*Porcellio scaber*	T	357	AJ388102
			Porcellio dispar	T	359	AJ388103
			Porcellio dilatatus dilatatus	T	353	AJ388104
			Porcellio dilatatus petiti	T	357	AJ388105
			Porcellionides cingendus	T	354	AJ388106
			Porcellionides pruinosus 1	T	348	AJ388107
			Porcellionides pruinosus 2	T	352	AJ388108
		Scleropactidae	*Scleropactes zeteki*	T	319	AJ388109

TAB. 7.1 – *Classification des 46 crustacés. T : espèce terrestre. A : espèce aquatique.*

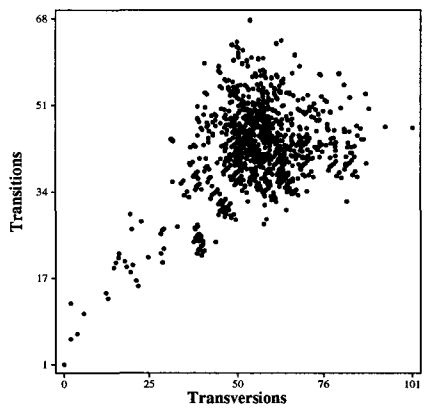


FIG. 7.1 – *Représentation de la saturation pour tous les couples.*

pas de travail préalable sur les séquences. Pour chaque couple de séquences, nous avons calculé directement la distance à partir des séquences brutes et ainsi produit une matrice de distances. Nous discutons dans les paragraphes suivants des paramètres utilisés pour la méthode de distance de transformation.

Distance brute ou taux de compression ? Jusqu'à maintenant, et particulièrement dans le chapitre précédent, nous avons presque toujours considéré les distances brutes, c'est-à-dire les distances obtenues par le calcul du poids du script. Tant qu'il s'agissait de comparer les résultats obtenus pour un même couple, l'utilisation de celle-ci était raisonnable. Si par contre nous voulons comparer des résultats obtenus pour des couples différents, si nous voulons pouvoir dire que telle séquence est plus proche d'une séquence que d'une autre, il est nécessaire d'utiliser une mesure relative et non plus absolue comme l'est le poids du script.

Obtenir une mesure relative se fait de manière tout à fait naturelle puisque nous nous situons dans un cadre de compression. La distance (brute) la plus grande que nous pouvons obtenir entre deux séquences est celle correspondant au script qui ne va pas utiliser d'information de S, c'est-à-dire au script contenant la seule opération '$\texttt{insertion}(T)$'. On définit donc la distance relative comme le rapport entre la distance brute obtenue et la distance brute correspondant au script 'insertion(T)' :

$$d_x(S,T) = \frac{d(S,T)}{w(\texttt{insertion}(T))}$$

La figure 7.2 montre bien que la distance brute ne permet pas de comparaison entre des couples différents. En effet, pour un même taux de compression, les valeurs des distances brutes sont très différentes. La distance brute serait utilisable pour les comparaison s'il y avait une

corrélation linéaire entre celle-ci et le taux de compression.

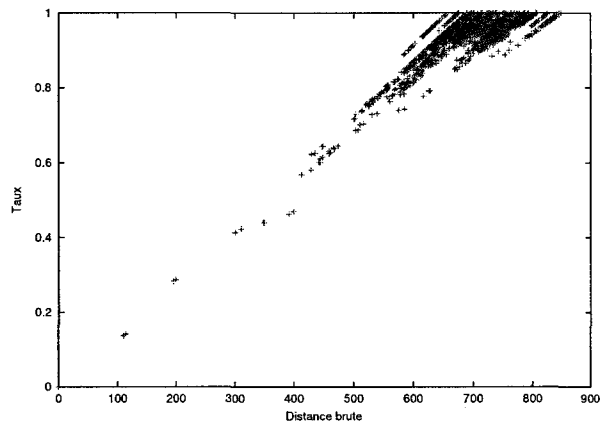


FIG. 7.2 – *Corrélation entre la distance brute et la distance par taux de compression. Le taux de compression est plus informatif et permet la comparaison des valeurs de distance pour des couples différents.*

Résolution de l'asymétrie. Nous avons évoqué au chapitre précédent le problème de l'asymétrie de la distance (section 6.2.2). Dans le cas d'une reconstruction phylogénétique il est nécessaire de disposer d'une mesure symétrique, nous choisissons donc $\frac{d(S,T)+d(T,S)}{2}$ comme mesure pour avoir cette propriété. De plus, la figure 7.3 révèle une corrélation quasi-linéaire entre la distance entre S et T et celle entre T et S.

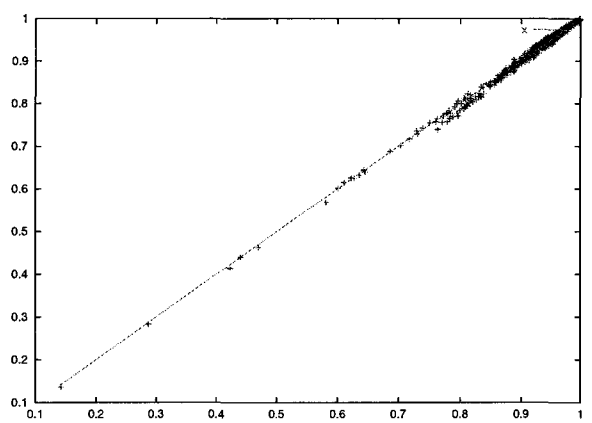


FIG. 7.3 – *Corrélation entre d(S,T) et d(T,S). On note une bonne symétrie sur cet échantillon de séquences.*

Le paramètre de longueur minimale des facteurs. Nous avons fixé le paramètre MFL à la valeur théorique que nous préconisons (voir 6.1.2), soit 5 pour la taille des séquences de l'échantillon. Tous les calculs que nous présentons n'ont été réalisés qu'avec des facteurs exacts.

Des expériences avec des facteurs avec substitutions ont été menées et n'ont pas permis d'obtenir de meilleurs résultats.

Fonction de poids et codage. Dans le chapitre 6, nous avons discuté des fonctions de poids. Nous avons mis en avant les bonnes propriétés de la fonction de poids w_2 qui présente notamment l'avantage de favoriser les séries de copies de facteurs faiblement écartés. C'est donc cette fonction de poids que nous utiliserons.

L'analyse des différents codages faite au chapitre précédent a révélé que le codage autodélimité avait tendance à produire des distances élevés, parce que peu de facteurs étaient sélectionnés. L'observation qualitative des scripts pour chaque couple de séquences confirme cela. L'analyse d'une phylogénie reconstruite à partir de la fonction de poids w_2 et de ce codage a montré que certaines séquences étaient à des positions qui ne sont pas conformes avec la réalité biologique. On note des couples d'espèces ne formant pas un seul groupe alors qu'il n'y a aucun doute sur leur groupement (*Janiridae* par exemple). Il existe également des positions surprenantes pour d'autres espèces (*Tylos europaeus*, *Platyarthrus hoffmanseggi* par exemple). D'autres arbres obtenus à partir des autres codages (le codage non-autodélimité et le codage information principale) résolvent bien ou donnent des positions congruentes à ces espèces par rapport aux analyses antérieures. Le codage non-autodélimité donne des distances moins grandes et repère plus de facteurs communs. Pour cette raison, il détecte mieux la similitude entre séquences.

Les distances obtenues avec le codage non-autodélimité sont plus réparties : la moyenne et la variance sont respectivement 0.93 et 0.007 pour le codage autodélimité, et 0.81 et 0.008 pour le codage non-autodélimité. Pour toutes ces raisons, nous avons donc choisi d'utiliser le codage non-autodélimité.

7.2.3 La reconstruction phylogénétique

Pour reconstruire les arbres phylogénétiques à partir de la matrice de distances produite par le calcul de la distance nous avons utilisé la méthode Neighbor-Joining [SN87].

Cette méthode s'inscrit dans les méthodes d'évolution minimale [DT93, Ber98]. Le principe est de rechercher une phylogénie dont la quantité d'évolution associée est minimale. On admet que la quantité d'évolution d'une phylogénie est représentée par la somme des longueurs des branches de celle-ci. La longueur d'une branche n'est pas mesurée comme la distance entre deux points mais évaluée aux moindres carrés. La calcul d'une phylogénie sur ce principe est supposé être NP-difficile. Une heuristique a donc été proposée par Saitou et Nei pour permettre un calcul approximatif d'une telle phylogénie : la méthode Neighbor-Joining [SN87].

Il s'agit d'un algorithme agglomératif, glouton, visant à regrouper à chaque étape deux espèces X et Y tel que la longueur de la phylogénie obtenue soit minimale. Il y a donc ajout à chaque étape de la branche la plus courte. Cependant, il n'est pas prouvé que cela conduise à la phylogénie la plus courte.

Pour s'assurer de la robustesse d'une phylogénie, on applique souvent une méthode de bootstrap. Celle-ci consiste à "reformer" un alignement en tirant au hasard et avec replacement des colonnes de l'alignement original, puis à recalculer la matrice de distances et l'arbre associé. On peut ainsi associer à chaque noeud un score suivant la proportion du nombre d'arbres dans lesquels ils apparaissent [Fel85]. Pour les arbres reconstruits avec la distance de transformation,

nous ne disposons pas de méthode similaire. Une manière de tester la robustesse des arbres est d'observer la sensibilité de ceux-ci à l'échantillonnage, c'est-à-dire aux espèces sur lesquelles l'arbre est reconstruit.

7.3 Résultats

7.3.1 La matrice de distances

Le calcul des distances indique une distance moyenne de 0.81 et une variance de 0.007. La distance minimale obtenue est de 0.085 pour le couple d'espèces *Artemia franciscana* / *Artemia salina*, et la distance maximale est 0.94 pour le couple *Artemia franciscana* / *Oritoniscus flavus*. 88% des distances se situent entre 0.71 et 0.91, 66% entre 0.76 et 0.86, et 54% sont supérieures à la moyenne.

7.3.2 Les phylogénies

Une expérience préliminaire a été menée avec l'ensemble de toutes les séquences, y compris les séquences problématiques révélées par l'analyse de Michel-Salzat et Bouchon. L'arbre reconstruit n'est pas montré. Les groupes externes *Artemia*, *Decapoda* et *Cumacea* sont effectivement placés à l'extérieur alors que les *Amphipoda* sont à la base des *Oniscidea*. Hormis les *Tylidae* et *Ligiidae*, les *Oniscidea* forment un seul groupe où les *Synocheta* (à l'exception de *Trichoniscus pusillus*) et *Crinocheta* sont regroupés. Les *Tylidae* et *Ligiidae* sont regroupés avec les isopodes aquatiques *Sphaeromatidae* et *Idoteidae* dont l'émergence semble antérieure aux *Janiridae*, et sont positionnés à la base des *Amphipoda* et autres *Oniscidea*.

Sachant que certaines espèces introduisent des biais dans la reconstruction phylogénétique (phénomène d'attraction des longues branches ou saturation en transitions/transversions), nous avons entrepris de reconstruire une phylogénie basée sur l'échantillonnage à partir duquel a été reconstruit l'arbre par des méthodes classiques. Nous aboutissons ainsi à un échantillonnage de 42 espèces où *Apseudes latreillei*, *Actaecia bipleura*, *Helleria brevicornis* et *Cyathura carinata* ont été retirées. L'arbre obtenu (figure 7.5) est presque identique au précédent. On note une réorganisation des isopodes aquatiques, et plus précisément des variations pour les familles *Tylidae* et *Ligiidae*. *Oniscus asellus* se place à la base des *Crinocheta*.

Afin de tester la robustesse de la phylogénie que nous avons reconstruite et de mieux apprécier les relations phylogénétiques entre les isopodes, nous avons effectué plusieurs reconstructions en enlevant à chaque fois un groupe externe supplémentaire. Nous avons ainsi réduit dans un premier temps l'échantillonnage aux malacostracés (40 espèces, *Artemia franciscana* et *Artemia salina* retirées, arbre non montré). Les espèces *Eurydice affinis*, *Tylos europaeus* (famille des *Tylidae*) et *Ligia oceanica* (famille des *Ligiidae*) voient leur positions modifiées. Les longueurs des branches des noeuds sur lesquels sont branchées ces espèces sont extrêmement petites. Nous avons ensuite réduit l'échantillonnage aux péracarides (38 espèces, *Penaeus japonicus* et *Penaeus monodon* retirées, arbre non montré). Encore une fois, les espèces représentatives des *Tylidae* et *Ligiidae* ont des positions différentes. Enfin, nous avons reconstruit un arbre à partir des seuls isopodes (35 espèces, *Cumopsis fagei*, *Gammarus locusta* et *Gammarus zaddachi* retirées, figure 7.8). Sur ces trois arbres, les *Amphipoda* restent à la base des *Oniscidea*. Les positions des *Tylidae* et *Ligiidae* sont variables mais occupent des positions à la base des arbres.

L'analyse a permis d'identifier quatre espèces pour lesquelles les positions étaient inattendues et différentes de celles proposées par Michel-Salzat et Bouchon: *Campecopea hirsuta*, *Asellus aquaticus*, *Eurydice affinis* et *Ligidum hypnorum*. Nous avons donc effectué une reconstruction à partir de l'échantillonnage à 42 espèces et où *Campecopea hirsuta* et *Asellus aquaticus* étaient retirées (40 espèces, figure 7.6), puis une reconstruction où *Eurydice affinis* et *Ligidum hypnorum* ont été également retirées (38 espèces, figure 7.7). Les *Amphipoda* sont alors placées comme groupe externe. De nouveau, les espèces représentatives de *Tylidae* et *Ligiidae* occupent des positions différentes de celles de l'arbre précédent. Les *Janiridae* rejoignent le groupe des isopodes aquatiques.

Sur l'ensemble des arbres reconstruits (figures 7.5, 7.6, 7.7 et 7.8) nous avons constaté que les positions des groupes suivants étaient conservées. Les espèces de la section des *Crinocheta* sont systématiquement regroupées. Dans cette famille, seules les espèces *Cylisticus convexus* et *Oniscus asellus* occupent des positions variables. Les sous-familles *Porcellionidae* et *Armadillidiidae* sont des groupes frères. Les deux espèces *Haplaphtalmus mengei* et *Oritoniscus flavus* de la section des *Synocheta* sont toujours groupées. Si l'on omet les deux *Gammaridae* souvent mal positionnés à la base des *Crinocheta*, les *Crinocheta* et les *Synocheta* sont deux groupes frères. Les *Sphaeromatidae* et les *Idoteidae* sont deux groupes frères. Les *Artemia* et les *Decapoda* sont toujours placées à l'extérieur.

7.4 Discussion

7.4.1 Analyse des distances

L'ensemble des phylogénies produites avec la distance de transformation révèle des branches assez longues. Cela provient de la répartition des distances calculées par notre méthode. La majorité des distances calculées sont concentrées sur un faible intervalle, proche de la valeur maximale que la distance de transformation peut atteindre, c'est-à-dire 1. Si l'on applique une fonction améliorant la répartition des distances à l'ensemble de la matrice, par exemple en élevant toutes les distances au carré, les longueurs des branches diminuent. De la même manière, si l'on applique une transformation réduisant l'espace de répartition des distances, par exemple la racine carrée, l'arbre reconstruit possède des branches plus longues. Les arbres présentés sont donnés sans cette transformation, celle-ci ne semblant diminuer que la longueur des branches sans modifier la topologie de l'arbre.

Comparativement à la matrice de distances obtenue avec la distance de Kimura à deux paramètres, les distances que nous obtenons sont effectivement réparties sur un intervalle plus petit. Cela signifie simplement que le critère de ressemblance détecté par la distance de transformation est plus sélectif.

Si l'on compare la valeur des distances que nous avons calculées à la longueur moyenne des facteurs sélectionnés dans le script minimal, à la somme des longueurs de ces facteurs et à leur nombre, il s'avère que la valeur de la distance de transformation sera d'autant plus petite que les facteurs sont grands et que leur nombre est petit. C'est-à-dire que les séquences évaluées les plus proches sont celles possédant de larges facteurs communs. Deux séquences pour lesquelles les facteurs communs sont courts mais nombreux seront évaluées plus distantes.

7.4.2 Analyse générale de la congruence avec l'arbre consensus

La comparaison de l'arbre de la figure 7.5, où les espèces problématiques pour la reconstruction classique ont été retirées de l'échantillon, et des arbres obtenus pour les trois sous-groupes malacostracés, péracarides et isopodes (seul l'arbre des isopodes est représenté figure 7.8), a permis de faire les constats suivants.

Les relations à la base entre les groupes externes *Artemia*, *Decapoda* et *Cumacea* sont respectées. Le quatrième groupe externe des *Amphipoda* est positionné avec les *Crinocheta* alors qu'il devrait se trouver à la base de l'arbre, voire comme groupe frère des isopodes [MSB00].

On constate la monophylie des *Crinocheta* avec des variations des positions pour les espèces *Oniscus asellus* et *Cylisticus convexus*. La monophylie des *Synocheta* n'est pas retrouvée, l'espèce *Trichoniscus pusillus* se rattachant systématiquement aux isopodes aquatiques. Les espèces des familles *Tylidae* et *Ligiidae* (marquées d'un point) occupent des positions variables mais sont systématiquement placées dans le groupe frère des *Crinocheta*.

Les positions des autres espèces sont assez variables, même si elles sont toutes majoritairement placées dans un groupe frère des *Crinocheta*. Notons que les *Idoteidae* forment un groupe frère des *Sphaeromatidae*, comme dans la phylogénie consensus. Les arbres obtenus pour les échantillons où les groupes externes sont enlevés sont congruents (figure 7.8).

L'analyse des arbres a révélé que quatre espèces étaient particulièrement mal placées : *Campecopea hirsuta*, *Asellus aquaticus*, *Eurydice affinis* et *Ligidum hypnorum*(soulignées par des tirets dans les arbres). Il s'avère que ces quatre espèces sont partiellement saturées et que l'on peut douter de leur placement dans l'arbre. Dans la phylogénie proposée par Michel-Salzat et Bouchon, celles-ci sont placées à la base des *Onicidea* (figure 7.4), *Campecopea hirsuta* et *Asellus aquaticus* formant un groupe frère de *Eurydice affinis* et *Ligidum hypnorum*. *Campecopea hirsuta* est particulièrement mal positionnée dans nos arbres puisqu'elle est située à la base de l'arbre, comme un groupe externe. Les trois espèces *Asellus aquaticus*, *Eurydice affinis* et *Ligidum hypnorum* sont quant à elles relativement proches mais regroupées avec les isopodes aquatiques. Le retrait de l'échantillon des deux espèces *Campecopea hirsuta* et *Asellus aquaticus* fait que les *Amphipoda* se placent bien comme groupe externe (figure 7.6). Le retrait de l'une ou l'autre de ces deux espèces n'induit pas ce repositionnement. Le retrait supplémentaire de *Eurydice affinis* et *Ligidum hypnorum* (figure 7.7) modifie la position de *Oniscus asellus*, la plaçant de manière identique à l'arbre consensus. Elle émerge ainsi avant *Scleropactes zeteki*, *Armadillo officinalis* et *Cubaris murina* alors que l'arbre avec 42 espèces (figure 7.5) proposait le contraire. Il est également intéressant de noter la position des *Janiridae*, positionnés à la base du groupe *Crinocheta* / *Synocheta* lorsque les quatre espèces *Campecopea hirsuta*, *Asellus aquaticus*, *Eurydice affinis* et *Ligidum hypnorum* font partie de l'échantillonnage (figure 7.5) et à la base des isopodes sinon (figures 7.6 et 7.7). Cette dernière position est congruente avec la phylogénie consensus proposée par Michel-Salzat et Bouchon.

7.4.3 Variation des positions des sous-familles des *Oniscidea*

La conclusion de l'étude menée par Michel-Salzat et Bouchon donnant les *Synocheta* et les *Crinocheta* comme groupes monophylétiques et frères n'est pas tout à fait confirmée ici. La monophylie des *Crinocheta* ainsi que leur apparition récente dans la phylogénie des isopodes est très bien soutenue par la reconstruction utilisant la distance de transformation. La position des *Synocheta* est dépendante de l'échantillonnage. Cependant, si l'on excepte les *Gammaridae*, les

Synocheta figurent comme groupe frère des *Crinocheta*.

Les trois espèces représentatives des *Synocheta* ne sont jamais regroupées, quel que soit l'échantillonnage. L'espèce *Trichoniscus pusillus* se rattache systématiquement aux isopodes aquatiques, à la base du groupe formé par les *Idoteidae* et les *Sphaeromatidae*. Le groupe formé des deux autres espèces représentatives de cette sous-famille (*Haplaphtalmus mengei* et *Oritoniscus flavus*) sont situées le plus souvent comme groupe frère des *Crinocheta*, sauf pour l'échantillon à 40 espèces représenté figure 7.6.

Pour les *Crinocheta*, on vérifie les résultats morphologiques et moléculaires que la sous-famille des *Porcellionidae* est un groupe frère de la sous-famille des *Armadillidiidae*, et qu'ils sont les groupes dérivés les plus récents. Il est intéressant de noter la position de *Cylisticus convexus* émergeant à la base des *Porcellionidae* et *Armadillidiidae* (des détails sur ce positionnement sont donnés dans la section suivante).

Les espèces représentatives des *Ligiidae* et *Tylidae* ont des positions très variables. On les retrouve tantôt regroupées avec les isopodes aquatiques *Idoteidae* et *Sphaeromatidae*, tantôt avec les *Crinocheta*. Cette grande variabilité ne permet pas de conclure. On note simplement que ces espèces ont des positions basales ce qui confirme l'hypothèse que ces groupes ont divergé assez tôt, avant les autres isopodes terrestres.

7.4.4 Congruence des arbres obtenus avec les différents échantillonnages

Position des *Amphipoda*

Nous avons noté que les *Amphipoda* étaient mal placées. On sait clairement qu'ils représentent un groupe externe et leur positionnement à l'intérieur de l'arbre est une erreur. Il est intéressant de remarquer que le retrait de *Campecopea hirsuta* implique le bon positionnement des *Amphipoda*, pour les échantillonnages des malacostracés et péracarides. Pour des échantillonnages plus grands, il est nécessaire de retirer en plus *Asellus aquaticus* pour que les *Amphipoda* reprennent leur position de groupe externe.

L'observation de l'alignement multiple (voir annexe A page 187) montre que les deux espèces d'*Amphipoda* partagent avec les *Isopoda* et en particulier avec les *Crinocheta* des portions supprimées aux mêmes positions alors que *Campecopea hirsuta* ne possède pas celles-ci. On remarque en observant la matrice de distances que les distances entre *Campecopea hirsuta* et les *Amphipoda* figurent parmi les quatre plus élevées et également que *Campecopea hirsuta* est plus proche de *Cumopsis fagei* ($d = 0.85$) que des *Amphipoda* ($d = 0.88$ et $d = 0.87$). Les distances avec les *Artemia* et les *Decapoda* sont également plus petites ($d = 0.83$, $d = 0.82$, $d = 0.81$, $d = 0.79$). Cela peut expliquer que *Campecopea hirsuta* se trouve rapprochée des groupes externes et avoir comme conséquence d'éloigner les *Amphipoda* de leur position externe.

Lorsque nous calculons la similitude entre deux séquences avec la distance de transformation, nous regardons globalement les séquences mais en détectant des ressemblances locales. Cela implique que nous trouvons, de manière qualitative, plus de ressemblances entre deux séquences qu'avec un alignement multiple.

Trichoniscus pusillus et les *Synocheta*

Parmi les *Synocheta*, l'espèce *Trichoniscus pusillus* n'est pas regroupée avec les autres espèces représentatives de cette famille, alors que cette espèce est une représentante type de la famille des *Synocheta*. De manière globale, *Trichoniscus pusillus* est placée à proximité des espèces *Asellus aquaticus* et *Ligidum hypnorum*.

La comparaison des scripts et des distances de transformation révèle une plus grande similarité entre *Trichoniscus pusillus* et ces deux dernières séquences. Les distances entre *Trichoniscus pusillus* et *Asellus aquaticus* et *Ligidum hypnorum* sont respectivement de 0.77 et 0.76 alors qu'avec les deux autres espèces de sa famille, *Oritoniscus flavus* et *Haplaphtalmus mengei*, les distances sont respectivement de 0.83 et 0.82. Etant donné la répartition des distances que nous avons (section 7.3.1), cela représente un écart assez important et signifie que nous évaluons une similarité effectivement plus grande. L'observation des scripts (voir figures 7.9 et 7.10) va tout à fait dans ce sens. On constate qu'il existe moins de facteurs communs, mais que ceux-ci sont globalement plus longs, entre *Trichoniscus pusillus* et *Ligidum hypnorum / Asellus aquaticus* (respectivement 22 facteurs couvrant 54% de la séquence et une longueur moyenne de facteurs de 9, et 30 facteurs couvrant 64% avec une longueur moyenne de 8) qu'entre *Trichoniscus pusillus* et *Oritoniscus flavus / Haplaphtalmus mengei* (respectivement 31 facteurs couvrant 60% et une longueur moyenne de 7, et 26 facteurs couvrant 50% de la séquence et une longueur moyenne de 7).

Par ailleurs, si l'on mesure la distance entre ces séquences grâce à la distance de Kimura (voir section 2.2.2), on trouve *Trichoniscus pusillus* plus proche de *Oritoniscus flavus* et *Haplaphtalmus mengei* lorsqu'on considère l'alignement épuré des zones d'indels, et plus proche de *Asellus aquaticus* et *Ligidum hypnorum* lorsqu'on considère l'alignement complet.

	TD	Kimura avec indels	Kimura sans indels
Haplaphtalmus mengei	0.82	0.61	0.41
Oritoniscus flavus	0.83	0.57	0.40
Asellus aquaticus	0.77	0.59	0.38
Ligidum hypnorum	0.76	0.49	0.35

TAB. 7.2 – *Comparaison des distances entre Trichoniscus pusillus et Haplaphtalmus mengei et Oritoniscus flavus d'une part et Asellus aquaticus et Ligidum hypnorum d'autre part pour la distance de transformation, la distance de Kimura sur les séquences complètes et la distance de Kimura sur les séquences épurées des indels.*

Il apparaît que la distance de transformation capte mieux la ressemblance entre les séquences puisqu'il existe effectivement plus de parties communes entre *Trichoniscus pusillus* et *Asellus aquaticus / Ligidum hypnorum* qu'entre *Trichoniscus pusillus* et *Haplaphtalmus mengei / Oritoniscus flavus*. Avec l'alignement final où certaines portions sont supprimées, la distance de Kimura montre l'inverse : *Trichoniscus pusillus* plus proche de *Oritoniscus flavus* et *Haplaphtalmus mengei*. Avec l'alignement complet, la distance de Kimura arrive à la même conclusion que la distance de transformation.

Il s'ensuit deux remarques. Premièrement, sans la correction manuelle de l'alignement, la distance de transformation et la distance de Kimura sont en accord. Deuxièmement, le signal phylogénétique est peut être dissocié de la similarité dans ce cas, ce qui implique une évolution particulière au sein de la section des *Synocheta*.

Position de *Oniscus asellus* et *Cylisticus convexus* parmi les *Crinocheta*

La comparaison des arbres obtenus avec les différents échantillonnages permet de conclure à la monophylie des *Crinocheta*, retrouvée à chaque fois. Cependant, deux espèces ont des positions variables suivant l'échantillonnage. *Oniscus asellus* a une position assez stable et émerge le plus souvent avant le groupe *Scleropactidae / Armadillidae*. *Cylisticus convexus* possède une position différente de celle proposée sur l'arbre de consensus, stable puisqu'elle occupe systématiquement cette position. Cette espèce émerge juste avant les groupes frères *Armadilliidiidae* et *Porcellionidae*.

Michel-Salzat et Bouchon précisent que la position qu'ils proposent, c'est-à-dire à l'intérieur du groupe des *Porcellionidae* (figure 7.4), est une erreur. La distance de transformation ne fait pas cette erreur. Ils indiquent également qu'une position à la base des *Porcellionidae* semble plus valide. La position que nous proposons, à la base des groupes frères *Armadilliidiidae* et *Porcellionidae*, semble aussi valide que cette dernière. Les données biologiques dont nous disposons actuellement ne permettent pas de trancher pour l'une ou l'autre des positions.

7.5 Résumé

Nous voulions confronter dans ce chapitre la distance de transformation à une véritable problématique biologique dans le but de valider notre méthode. Nous nous sommes intéressés à la phylogénie des isopodes terrestres, à leur position au sein des isopodes, ainsi qu'à la position des sous-groupes. Plusieurs difficultés ont suscité l'intérêt d'utiliser la distance de transformation dans le contexte de cette analyse. La difficulté à produire un alignement multiple a été bien sûr une raison majeure. L'observation de blocs complets supprimés et la suspicion de mouvements de blocs dus à la structure secondaire des ARN faisait de ces séquences un sujet d'étude intéressant pour notre mesure. Il y avait notamment un intérêt à disposer d'une méthode capable de produire une phylogénie directement à partir des séquences, sans alignement préalable. D'autre part, le peu de consensus sur la phylogénie des isopodes fait que l'observation des arbres obtenus par l'application de méthodes différentes est un moyen de soutenir ou d'infirmer certaines hypothèses.

Nous retrouvons bien la monophylie des *Crinocheta*. Les positions des familles au sein de cette section sont bien supportées, quel que soit l'échantillonnage de séquences. De plus les relations proposées sont congruentes avec les données connues. Deux séquences, *Cylisticus convexus* et *Oniscus asellus*, occupent des positions différentes de celles obtenues par des analyses classiques. On ne dispose pas de connaissances suffisantes pour affirmer que ces positions sont valides ou non. Dans le cas de *Cylisticus convexus*, la distance de transformation évite une erreur de positionnement par rapport à l'analyse de [MSB00]. Il semble également que cette position ne soit pas moins valide qu'une autre. Les *Armadilliidiidae* et les *Porcellionidae* sont bien retrouvées comme groupes frères et correspondent à l'évolution la plus récente au sein des isopodes.

La séquence *Trichoniscus pusillus*, qui n'est pas regroupée avec les *Synocheta*, ne nous permet pas de conclure à la monophylie de cette section. L'étude des scripts a révélé une réelle ressemblance entre cette séquence et celles avec lesquelles elle était regroupée. Les deux autres espèces représentatives des *Synocheta* sont, comme attendu par Michel-Salzat et Bouchon, positionnées comme groupe frère des *Crinocheta*.

Les positions des deux familles des *Tylidae* et *Ligiidae* ne sont pas plus résolues qu'avec des études phylogénétiques classiques. Il est simplement possible de constater que ces groupes ont

dû émergé assez tôt, avant les autres isopodes terrestres.

La congruence des positions de certains groupes par rapport à des phylogénies reconstruites à partir de méthodes classiques permet d'affirmer la capacité de la distance de transformation à capter les ressemblances entre séquences. Les différences constatées doivent poser la question de la pertinence de ce que nous proposons. Du point de vue de la détection de la ressemblance, l'observation des scripts permet de conclure que nous détectons une plus grande quantité d'informations partagées. La validité de cette ressemblance n'est pas à remettre en cause si nous étudions les séquences pour évaluer en quoi elles se ressemblent. L'utilisation de notre mesure à des fins phylogénétiques demande sans doute de travailler sur un ensemble de facteurs communs significatifs du point de vue de la biologie.

Dans des contextes où l'on dispose d'un ensemble de séquences peu mutées, où l'alignement ne pose pas de difficulté majeure, l'utilisation des méthodes traditionnelles fournit des résultats satisfaisants. Il sera intéressant d'utiliser la distance de transformation pour les cas où les données ne peuvent être analysées par les méthodes usuelles. Dans le cas de séquences courtes, il s'agira le plus souvent de données à partir desquelles il est difficile, voire impossible, de produire un alignement satisfaisant. Nous avions traité dans [Var96, VDR99] des séquences du rétrotransposon Tnt1 du tabac [CVG95]. Celles-ci avaient la particularité de disposer de blocs similaires communs en nombre variable. Une analyse par alignement ne permet pas de détecter de telles ressemblances. Dans le cas de séquences plus longues, des chromosomes complets par exemple, les méthodes usuelles n'ont pas de sens. Il est alors nécessaire de concevoir de nouvelles méthodes, la distance de transformation en constitue une.

Figures

Afin d'obtenir une représentation plus claire, certaines familles, groupes ou espèces ont été marqués sur les arbres :

- les groupes externes sont marqués d'un 'E',
- les espèces soulignés de tirets sont celles ayant des positions inattendues et différentes de l'arbre consensus (*Campecopea hirsuta*, *Asellus aquaticus*, *Eurydice affinis* et *Ligidum hypnorum*),
- les branches marquées d'un point indiquent les espèces des familles *Tylidae* et *Ligiidae*,
- les groupes conservés les plus importants sont notés.

Les figures 7.9 et 7.10 représentent des scripts obtenus pour des couples de séquences. Les séquences sont extraites de l'alignement afin de mieux comparer les ressemblances obtenues par la distance de transformation et par l'alignement. Les schémas doivent être lus ainsi :

- la première ligne représente la règle,
- les séquences sont ensuite données par blocs. Les segments communs sont nommés, de 'a' à 'Z'. Ainsi, 'aaa' représente un segment de longueur 3 d'un facteur. Chaque bloc est composé de 5 lignes :
  1. la séquence cible,
  2. le nom des segments communs dans la cible,
  3. le nom des segments communs dans la source,
  4. la séquence source,
  5. une suite d'étoiles indiquant là où deux bases sont identiques.

Il est possible que tout ou partie d'un segment d'un facteur soit caché dans la source puisque les chevauchements sont autorisés et que la représentation ne prend pas en compte cela. On retrouve également parfois des blocs d'indels alignés, cela provient de l'utilisation des séquences représentées comme dans l'alignement multiple.

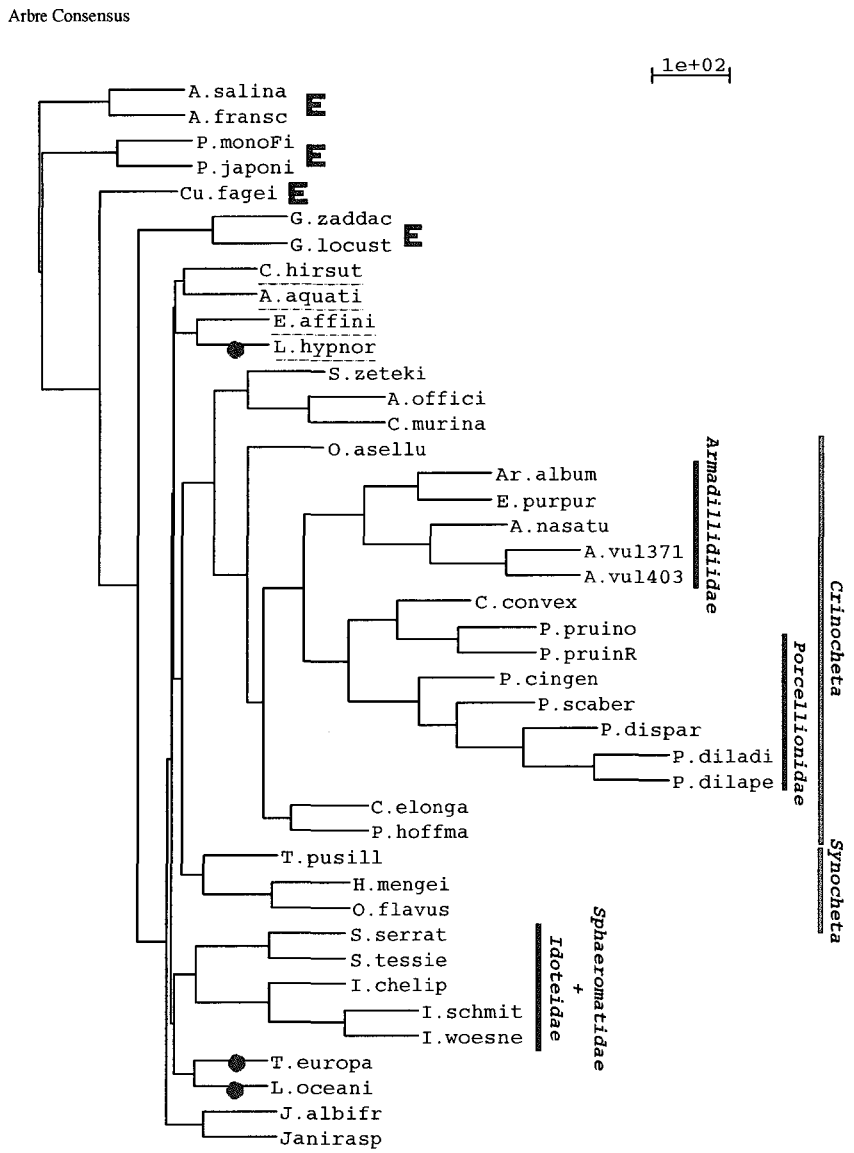


FIG. 7.4 – *Arbre de consensus obtenu à partir de 1000 tests de bootstrap sur 42 crustacés (4 espèces problématiques retirées) [MSB00]*

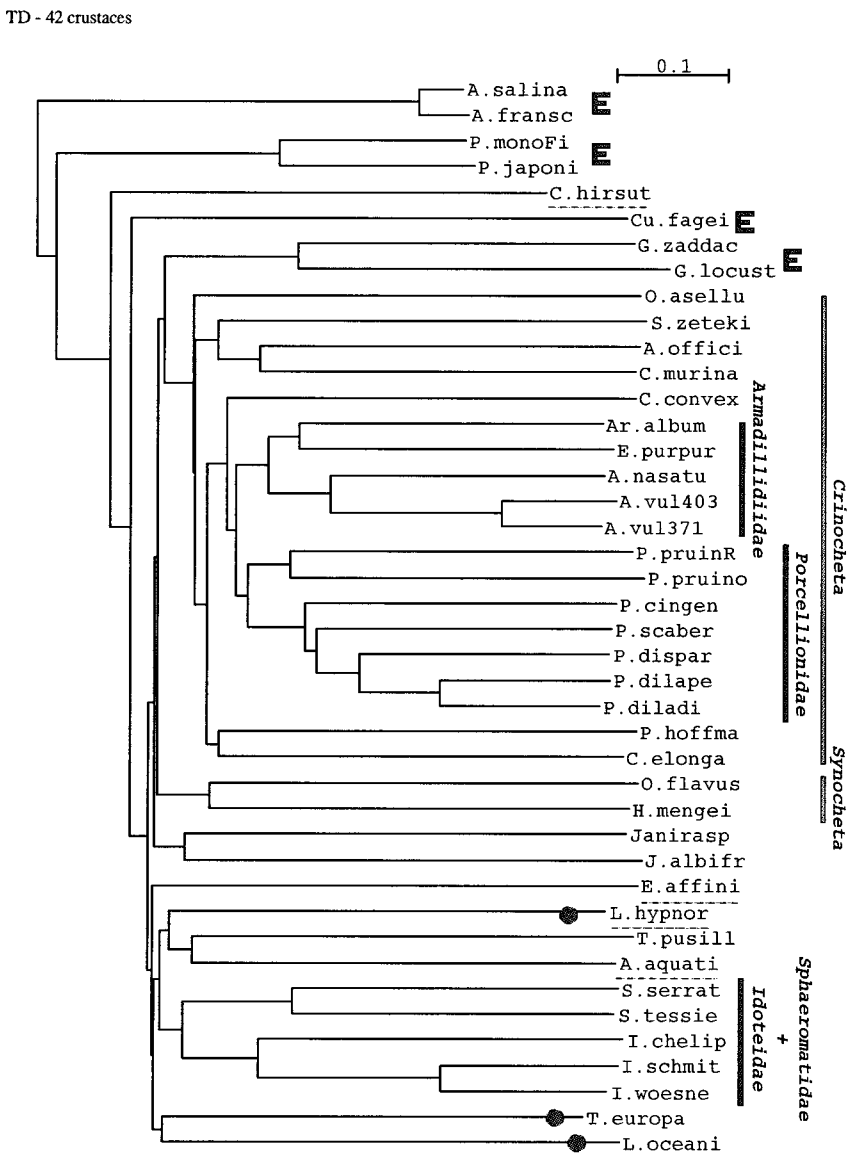


FIG. 7.5 – *Arbre reconstruit avec la distance de transformation, 42 espèces de crustacés (Apseudes latreillei, Actaecia bipleura, Helleria brevicornis et Cyathura carinata retirées).*

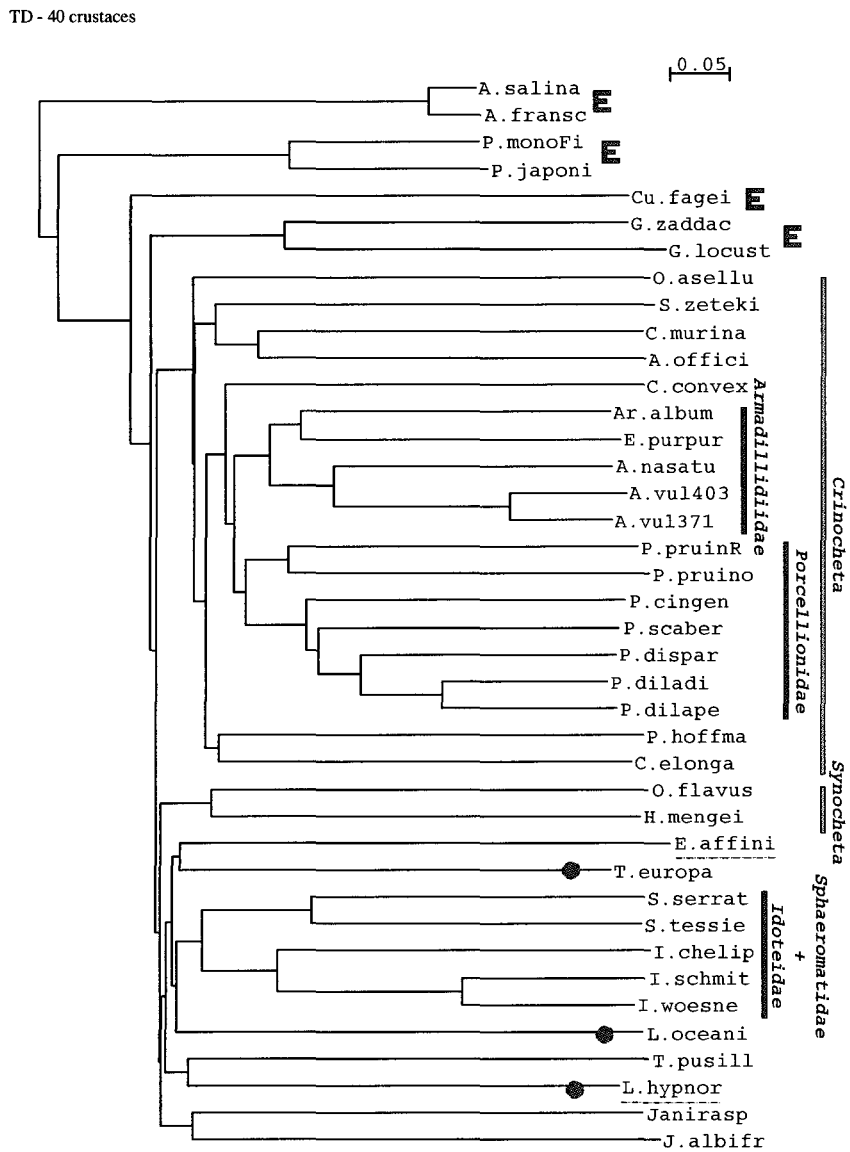


FIG. 7.6 – *Arbre reconstruit avec la distance de transformation, 40 espèces de crustacés (Campecopea hirsuta et Asellus aquaticus retirées).*

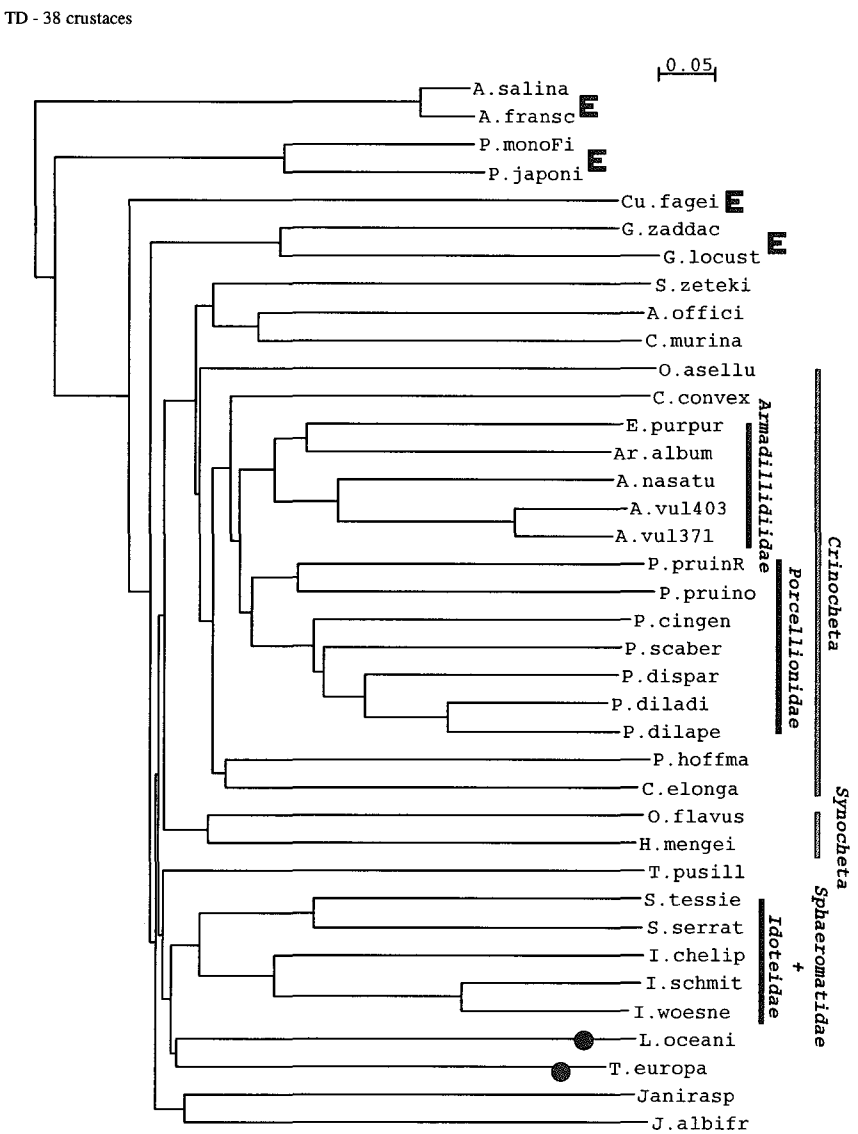


FIG. 7.7 – *Arbre reconstruit avec la distance de transformation, 38 espèces de crustacés (Campecopea hirsuta, Asellus aquaticus, Eurydice affinis et Ligidum hypnorum retirées).*

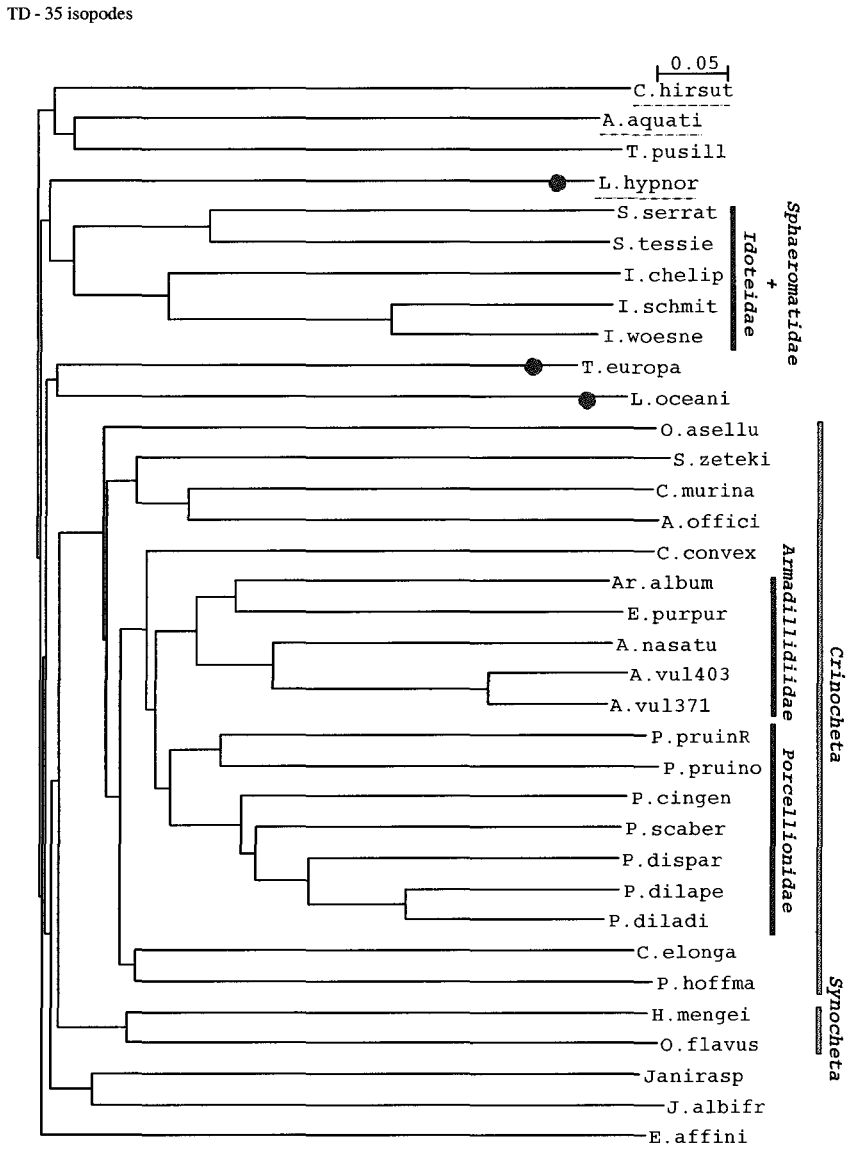


FIG. 7.8 – *Arbre reconstruit avec la distance de transformation, 35 espèces d'isopodes.*

FIG. 7.9 – *Comparaison de Trichoniscus pusillus et des deux autres espèces représentatives des Synocheta : Oritoniscus flavus et Haplaphtalmus mengei.*

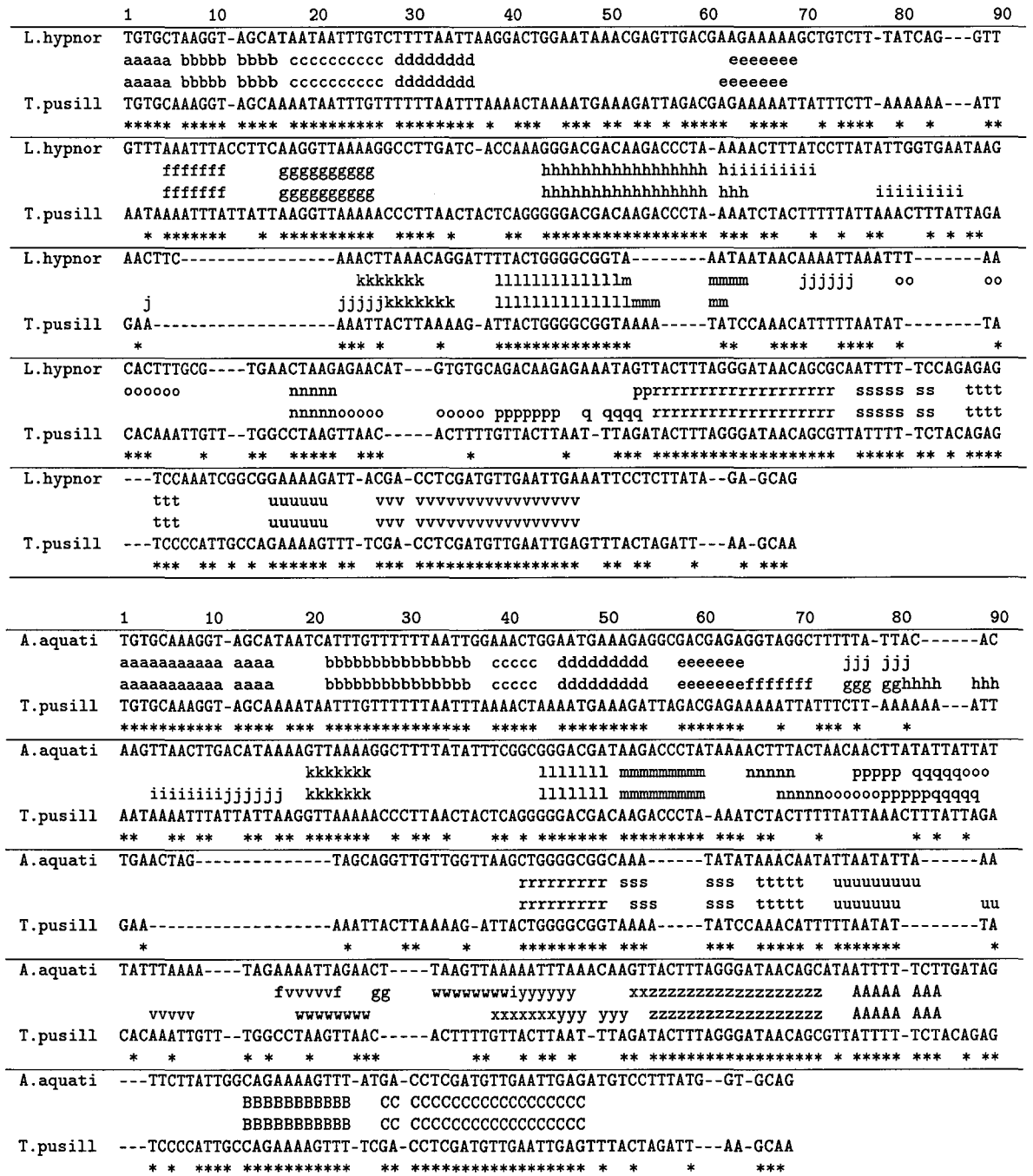


FIG. 7.10 – *Comparaison de Trichoniscus pusillus avec les séquences avec lesquelles elle est positionnée dans l'arbre : Ligidum hypnorum et Asellus aquaticus.*

Conclusions

Nous avons présenté dans ce travail une nouvelle notion de distance entre séquences génétiques. Les fondements théoriques sur lesquels repose la définition de cette distance en font un nouvel outil d'identification de la ressemblance. La distance de transformation présente plusieurs avantages par rapport aux méthodes de comparaison traditionnelles. Elle s'appuie sur l'identification de segments communs et offre ainsi une vision plus globale de la ressemblance que l'étude par des mutations ponctuelles. Les fonctions de poids des opérations sont définies relativement à l'information contenue dans ces opérations, et ne sont pas données de manière *ad hoc*. L'espace de recherche du script minimal parmi l'ensemble des scripts possibles permet d'envisager un grand nombre de schémas de ressemblance, du schéma classique d'alignement (où les facteurs ne se croisent pas) au schéma envisageant une complète recomposition des séquences. Cela rend la distance de transformation particulièrement utile lorsqu'on dispose de séquences qui ne permettent pas de construire un alignement.

Dans un souci constant de rendre utilisable notre travail par les biologistes, nous avons cherché à disposer d'algorithmes rapides du calcul de la distance de transformation. L'étude approfondie des scripts non optimaux a permis de dégager des propriétés réduisant la taille du problème à traiter. L'heuristique du graphe de scripts compact permet alors un calcul très rapide de la distance avec une très bonne approximation. On réservera l'heuristique du graphe de scripts valides au traitement de données très spécifiques, pour lesquelles on voudra par exemple forcer la prise en compte de certains facteurs communs. Grâce à l'étude que nous avons réalisée au chapitre 6, nous avons acquis une bonne connaissance de la distance, de ses propriétés et de la ressemblance qu'elle détectait.

Les poursuites que l'on peut associer à ce travail sont nombreuses.

L'utilisation de la distance de transformation sur divers échantillons de séquences a révélé certaines limites relatives à la définition des fonctions de poids en se basant strictement sur de critères informationnels. La qualité de la ressemblance mesurée par notre méthode serait accrue si l'on réussissait à déterminer de manière plus précise le poids à accorder aux opérations, en particulier l'importance du ratio poids d'une insertion / poids d'une copie, en s'appuyant sur des données biologiques. D'autre part, il serait intéressant de distinguer deux types d'insertions et de leur associer des poids différents : l'une consacrée à l'insertion de longs segments, l'autre permettant d'insérer les petites parties inter-copies. Un poids plus faible accordé à la seconde permettrait de favoriser encore les suites de petits facteurs communs.

Dans le cadre de l'étude d'ensembles d'espèces à des fins de reconstruction phylogénétique, il serait intéressant de proposer une extension de la distance qui permette de prendre en compte les ressemblances communes à plusieurs séquences. Plusieurs voies sont envisageables. On peut imaginer disposer d'un ensemble de facteurs communs calculé à partir de l'ensemble de toutes les

séquences de l'échantillon. Un score de multiplicité serait alors associé à chaque facteur commun. Les fonctions de poids pourraient être définies en tenant compte de ce critère. Un facteur commun à toutes les séquences représente un signal phylogénétique moins fort qu'un facteur commun à deux séquences lorsqu'on estime la similarité entre deux séquences données. La prise en compte de ce critère dans les fonctions de poids aurait pour conséquence d'accorder plus d'importance à des ressemblances locales à certains sous-groupe, voire certains couples.

Les diverses expériences menées au cours de ce travail ont permis de mesurer la difficulté qu'il y avait à concevoir une distance applicable à n'importe quel type de séquences, qu'elle soit longue ou courte, codante ou non codante. La spécification de la distance de transformation pour des types de données apparaît nécessaire. Le traitement de courtes séquences (ARN ribosomiques, gènes, etc) peut être réalisé grâce à la méthode que nous avons implémenté. La faible longueur des séquences rend la détection des parties communes réalisable. Pour l'application à de plus longues séquences (chromosomes, génomes complets) il est nécessaire d'envisager un autre mode de détection des segments communs. Bien que notre méthode soit capables de traiter de longues séquences, il semble préférable que l'ensemble des facteurs communs soit donné : soit par le biologiste, qui possède l'expertise nécessaire à la comparaison des séquences traitées, soit par l'intermédiaire de programmes tels que BLAST, capables de détecter des similarités locales, ou encore en utilisant les annotations associées aux séquences. Les fonctions de poids doivent être également adaptées pour prendre en compte les mutations ponctuelles, qu'il est impossible de ne pas autoriser dès lors que l'on traite des séquences assez longues.

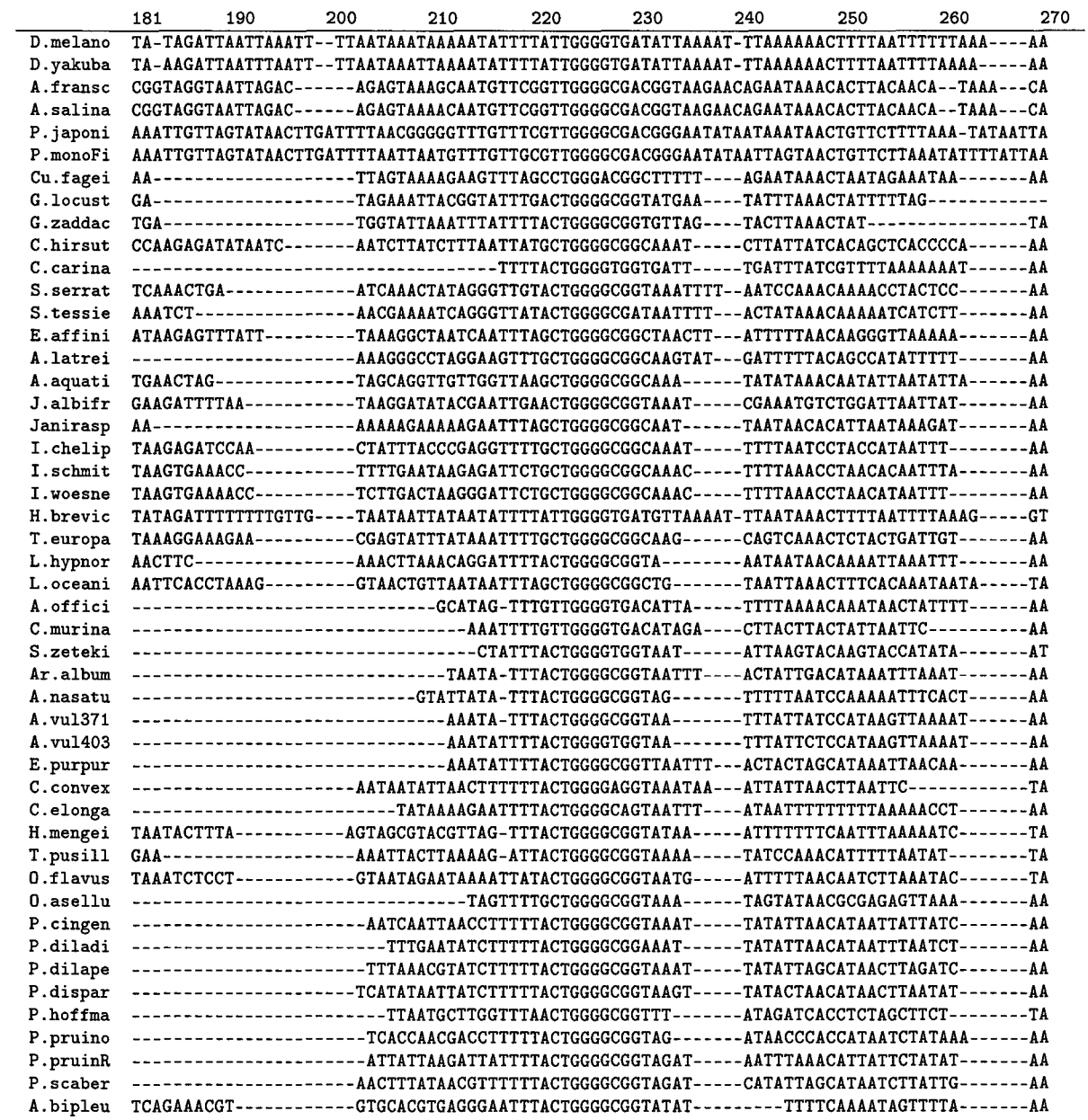
Comme il apparaît, les perspectives de poursuite de ce travail ne manquent pas. La comparaison de séquences est toujours d'actualité dans le domaine de la bioinformatique, et des travaux utilisant la théorie de l'information abordent le problème avec un regard différent [APD99]. La mise à la disposition de génomes complets ouvre la voie à de nombreuses applications où nous espérons que la distance de transformation jouera un rôle.

Annexe A

Alignement des séquences Isopodes

```
            1        10        20        30        40        50        60        70        80        90
D.melano    TGTGCAAAGGTTAGCATAATCATTAGTCTTTTAATTGAAGGCTGGAATGAATGGTTGGACGAAATATTAACTGTTTC-ATTTAAAATTTT
D.yakuba    TGTGCAAAGGT-AGCATAATCATTAGTCTTTTAATTGAAGGCTGGAATGAATGGTTGGACGAAATATTAACTGTTTC-ATTTAAATTTAA
A.fransc    CATGCGAAGGTTAGCATAATCATTAGCCTTTTGATTTGAGGCTGGAATGAATGGTTTGACGAGAGATGGTCTGTCTC-TTCGATT---AA
A.salina    CATGCGAAGGT-AGCATAATCATTAGCCTTTTGATTTGAGGCTGGAATGAATGGTTTGACGAGAGATGGTCTGTCTC-TTCGATT---AA
P.japoni    CGTGCGAAGGT-AGCATAATCATTAGTCTTTTAATTGGAGGCTTGTATGAATGGTTGGACAAAAAGTAAGCTGTCTC-GATTATA---AT
P.monoFi    CGTGCGAAGGT-AGCATAATCATTAGTCTTTTAATTGAAGGCTTGTATGAATGGTTGGACAAAAAGTAATCTGTCTC-AGTTATA---AT
Cu.fagei    CGTACTAAGGT-AGCATAATCCTTCGTTATTAATTTTAAACGTAGAATGAAAGGTATGACGTAGTGGACTCTTTAAC-TTTCTAT----A
G.locust    TGTGCTAAGGT-AGCATAATCATTTGTTTCTTAATTGGTAACTGGCATGAAAGGTTGAACCAATTTTACCGGTTTGT-GTGCTTT---TT
G.zaddac    TGTGCTAAGGT-AGCATAATAATTTGCTATTTAATTGGTAGCTGGAATGAAAGGTTGAACTAATTTTATTTATTTGC-TTAACAA---AA
C.hirsut    TGTGCCAAGGT-AGCATAATCATTAGCCTTTTAATTGAAGGTTGGTATGAATGAATCGACGAGAGAGAGCTCTTTTTTAACCTATTCTTA
C.carina    TGTACAAAGGT-AGCAAAGTAGGTAGTTTAATAATTTTAAACCAGAATAAATGATTAAACGGGAAATCGGTTGTCCTCAGTTGGAAATTA
S.serrat    TGTGCTAAGGT-AGCATAATAATTAGCTTCTTCATTGGAAGCTCGTATGAACGACCCGACGAAAAATCCCCTGTCTC-AACAAT----CC
S.tessie    TGTGCTAAGGT-AGCATAATAATTAGCTTTTTAATTGGAAGCTCGAATGAACGACCCGCCGAAAAATCTCCTGTCTC-AGCGCT----CC
E.affini    CGTGCTAAGGT-AGCACAATCACTTGCTTTTTTATTGAAAGCCAGAATGAAAGTGTTGACATAGAACAGCTGACTCC-TTTTTA----CC
A.latrei    TGTGCTAAGGT-AGCATAATCATTTGGCTTTTAATTAAGGTCTCGTATGAAAGGTTTAACATAACTTTAAACGTCTC-CCCCTC----AA
A.aquati    TGTGCAAAGGT-AGCATAATCATTTGTTTTTTAATTGGAAACTGGAATGAAAGAGGCGACGAGAGGTAGGCTTTTTA-TTAC------AC
J.albifr    TGTGCTAAGGT-AGCATAATAATTTGTCCTATAATAGAAGGCTGGAATGAATGAGTTGACGATAGATATACTTTATC-CTAAGGTGAGTA
Janirasp    TGTGCTAAGGT-AGCATAATAAATTGTCTTATAATTAAAGACTAGAATAAATGGGTTGACAATCTTTTTACTTTTTT-AATTTG----AA
I.chelip    TGTGCTAAGGT-AGCATAATCATTTGTCTCCTGATTAGAGACTAGTATGAATGGTTAGACGAGGGGAATTCTTTCTT-AACTAA---ATA
I.schmit    TGTGCTAAGGT-AGCATAATCACTTGTCTTCTAATTAGAGGCTGGAATGAATGGTATGATGAGAGAAGAGCTTTCTT-AATTAA---ATT
I.woesne    TGTGCTAAGGT-AGCATAATCACTTGTCTTCTAATTAAAGGCTGGAATGAATGGTACGACGAGAGAAGAGCTTTCTT-AATTAA---ATT
H.brevic    TGTGCAAAGGT-AGCCTAATCATTAGTCTTTTAATTGAAGGCTGGTATGAATGGTTGGCCGAGATATTAACTGTTTCATAAAAATTTATA
T.europa    TGTGCTAAGGT-AGCATAATAATTTGCTTTTTAATTGAAAGCTGGAATGAAAGATGTGACGAGAAGATAACTGTCTT-GGTTTA---TTA
L.hypnor    TGTGCTAAGGT-AGCATAATAATTTGTCTTTTAATTAAGGACTGGAATAAACGAGTTGACGAAGAAAAAGCTGTCTT-TATCAG---GTT
L.oceani    TGTGCTAAGGT-AGCATAATAATTTGCCTCGTAATTGGGGGCTAGAATGAAGGATCTGACGAAGGAAACCCTGTCTT-GACATT---TGA
A.offici    TGTGCTAAGGT-AGCATAATCATTTGTTATCTAAATGATAACTAGTATGAATGGTCTGACGAGAAAATAACTTTCTA-AAAAGA--AATA
C.murina    TGTGCTAAGGT-AGCATAATAATTTGTTATTTAATTGATAACTGGCATGAAAGGTCAAACG-GAAAATATCTTTCTC-TTTATT---TCA
S.zeteki    TGTGCTAAGGT-AGCATAATCCTTTGTTATTTAATTAATATCTTGAATGAAGGGTTTGACGAGAGAGAGACTTTTTT-AAATA----GAT
Ar.album    TGTGCTAAGGT-AGCATAATAATTAGTTATTCAATTGGTAACTAGAATGAATGGATTGACAAAAAAATAACTTTATT-AATAA-----TT
A.nasatu    TGTGCTAAGGT-AGCATAATAATTAGCTATTTAATTTGTAGCTGGAATGAATGGTCTGACAAGATAATAACTTTCTT-ATTTC-----TG
A.vul371    TGTGCTAAGGT-AGCATAATCATTAGTTACTTAATTTGTAACTAGAATGAACGGCCTGACAAAAGAATAACTTTATT-ATTTT-----TT
A.vul403    TGTGCTAAGGT-AGCATAATCATTAGTTACTTAATTTGTAACTAGAATGAACGGCCCGACAAAAGAATAACTTTATT-ATTTTT----TT
E.purpur    TGTGCTAAGGT-AGCATAATAATTAGCTACCTAATTAGTAGCTAGAATGAACGGTCGGACAAAAAACTTACTTTATT-TTTTC-----AC
C.convex    TGTGCTAAGGT-AGCATAATAATTAGTTGCTTGATTAGCTACTGGAATGAAAGGCCTGACGAGAGATAACCTCTCTC-TTATT-----TT
C.elonga    TGTGCTAAGGT-AGCATAATCATTTGATACTTAATTGGTATCTGGAATAAAAGACCTGACGAAAAGTTAACTTTCTT-TTTCT----AGC
H.mengei    TGTGCTAAGGT-AGCATAATCACTAGTCTATTAATTGTAAACTCGAATGAACGATTTGACGAGAAAACTCCTGTCTT-TATTT----AAT
T.pusill    TGTGCAAAGGT-AGCAAAATAATTTGTTTTTTAATTTAAAACTAAAATGAAAGATTAGACGAGAAAAATTATTTCTT-AAAAAA---ATT
O.flavus    TGTGCTAAGGT-AGCATAATAATTTGGTGTTTAATTAACATCTGGTATAAACGATCTGACGAAAATTTAGCTTTCTC-CTATAA---AAG
O.asellu    TGTGCTAAGGT-AGCATAATCATTTGCTACTTAATTAGTGGCTAGAATGAATGGGAGGACGAATAGTTGACTTTCTT-AGTTT-----AC
P.cingen    TGTGCTAAGGT-AGCATAATAATTAGTTATTTAATTGATAACTTGGATGAAAGGTTTGACAAAAAGAAGATTTTCTT-TTTAT-----TT
P.diladi    TGTGCTAAGGT-AGCATAATAATTAGTTATTTAATTGATAACTAGAATGAAAGGTTTGACGCAGAGAAAAATCTTCTT-ATTAT-----AA
P.dilape    TGTGCTAAGGT-AGCATAATAATTAGTTATTTAATTGATAACTAGAATGAAAGGTTTGACGCAAAGAAAATCTTCTT-GTTAT-----AA
P.dispar    TGTGCTAAGGT-AGCATAATAATTAGTTATTTAATTGATAACTAGAATGAAAGGTCTGACGAAAAGAAAATCTTCTT-TTTAT-----AA
P.hoffma    TGTGCTAAGGT-AGCATAATAATTTGCTGTTTAATTGGCAGCGAGCATAAAAGATCTGACGAAAGTCAAACTTTTTT-TCTGT-----CC
P.pruino    TGTGCTAAGGT-AGCTTAATAATTAGCAACCTAATTAGTTGCCGGTATAAATGGTCTGACGAAAAATTACTTTTCTT-TGGTT-----CC
P.pruinR    TGTGCTAAGGT-AGCATAATAATTAGTAGCCTAATTAGCCACCAGAATGAATGACCTGACGAAAAATAAATTTTCTT-GTATT-----TT
P.scaber    TGTGCTAAGGT-AGCATAATAATTAGTTATTTGATTGATAACTTGAATGAAAGGTTTGACGAAAAGAAAATCTTCTC-TCTACG----AA
A.bipleu    TGTGCAAAGGT-AGCATAATCATTAGTTCTTTAATTGAGGAATGGAATGAAGGATTTGACGAGGTCTGAACTATCTA-ATTTTTA--AGA
```

```
            91        100       110       120       130       140       150       160       170       180
D.melano    TATAGAATTTTATTTTTTAGTCAAAAAGCTAAAATTTATTTAAAAGACGAGAAGACCCTATAAATCTTTATATTTTTTTTATTTTAATTA
D.yakuba    AATAGAATTTTATTTTTTAGTCAAAAAGCTAAAATTAATTTAAAAGACGAGAAGACCCTATAAATCTTTATATTTTATTTATTTTAATTA
A.fransc    ATTGAAGTT-AATCTTTAAGTGAAAAAGCTTAAATGTACTTGGAGGGCGATAAGACCCTATAGATCTTTACATTTAATTCTTTTGTCTTG
A.salina    ATTGAAGTT-AATCTTTAAGTGAAAAAGCTTAAATGTACTTGGAGGGCGATAAGACCCTATAGATCTTTACATTTAATTCTTTTGTCTTG
P.japoni    AATTGAACTTAACTTTTAAGTGAAAAGGCTTAAATGTTTCAGG-GGACGATAAGACCCTATAAAGCTTGACAATAACTTCGTTATATTAT
P.monoFi    AGTTGAACTTAACTTTTAAGTGAAAAGGCTTAAATACTTTAAGGGGACGATAAGACCCTATAAAACTTAACAATAATTTGATTAAATTAT
Cu.fagei    AGAATAATT--TTCTTTAAGGTAAAAATGCCTTAATTAAGGGAAGGACGATAAGACCCTA-AAAATTTAACTTCCTTTTTTTGCCCTATG
G.locust    TAGGAAATT--TAAGTCAAGGTAAGAATTCTTTGATCTCCTAGGGGACGATAAGACCCTA-GAAGCTTTATACGTTGTATAACTTTTAAT
G.zaddac    AAAGAAATT--AAAGTTGAGGTAAGAATTCTTTAATTTTCTAGGGGACGATAAGACCCTA-GAAGCTTGATAAATATAAATAATTATTAG
C.hirsut    AAAAATTTAATGACAAAAGTGAAAACACTTTTTTTTATTTAGAAAGACGATAAGACCCTATAAAACTTTATTAATAAATAAGTTTAAATA
C.carina    AAAGGTTTATTTTAAGGATTTCAAAAGGCCTTTTTTTTTTAAGAAGACGATAAGACCCTA-GAAGCTAAAAA----------------
S.serrat    AGATTAACTTTCTAAGAGAGTCAAAAAGCTCTCCTTACCTAATAGGACGACAAGACCCTATAAAACTTTACCTTTTAAGCTTACACTCAT
S.tessie    -TATTAACTTACTTACAGAGTAAAAACGCTCTCTTTACTTAAGAGGACGACAAGACCCTATAAAACTTTACCCTATTATCGGCCTACTTA
E.affini    TTTAAATCT-AAGCGTAAGGTTAAAACGCCTTAAATCTCTAGAAGGACGACAAGACCCTATAAAACTTTATTGGTCCAGCCTTAACCTAT
A.latrei    ATCCTAAGTTAGTATCTAGGTGATAAACCCCAGATAACCCAGAGGGACGACAAGACCCTA-AAAGCTATTTTATTTCGGGTGGACAAGGT
A.aquati    AAGTTAACTTGACATAAAAGTTAAAAGGCTTTTATATTTCGGCGGGACGATAAGACCCTATAAAACTTTACTAACAACTTATATTATTAT
J.albifr    TTTGAATTTAATATGTAAATAAGAAAGTTTAGTTAATATTAGTGGGACGATAAGACCCTGTGAAATTTAATTTGTAGTGTTGTTTTTTTA
Janirasp    TCTTGAAATTTATTTTTAAGTCAAAAAGCTTAATTTTTTTAGGTAGACGATAAGACCCTGTAAAACTTTATTTTTAATAAATAGTAAGAA
I.chelip    AACTAAAATTAGCTTTAAAGTCAAAAGCCTTTAAGAACTTAGGGGGACGACAAGACCCTATAAAACTTTACCTCCTGTAAAATTTAATCC
I.schmit    TAAAGAACTTATCTTGGAAGTTAAAAGACTTCCAAAATTTAGGGGGACGACAAGACCCTATAAAACTTTATCCCTTTTTCAACTTAAGCT
I.woesne    TTAAAAACTTATCTTAGAAGTCAAAAGACTTTTAAAATTTAGGGGGACGACAAGACCCTATAAAACTTTATCCCCTTTTCAACTTAAGAT
H.brevic    ATATGAATTTTATTTTTTAGTTAAAAAGCTAAAATATATTTAAAAGACGAGAAGACCCTATAAATCTTTATATTTAAGTTATTATAATTT
T.europa    AAAATAACTTGACTGAGAAGTAAAAAGGCTTCTATGGTTTTAAGGGACGATAAGACCCTATAAAGCTTTATTTATAATTACTGGTTTATT
L.hypnor    GTTTAAATTTACCTTCAAGGTTAAAAGGCCTTGATC-ACCAAAGGGACGACAAGACCCTA-AAAACTTTATCCTTATATTGGTGAATAAG
L.oceani    CCTAAAATTTAATATTTAAGTAAAAAAGCTTAGATTAAGCGAAGGGACGACAAGGCCCTA-AAAGCTTTTTTATTGACGTAAATCTGATA
A.offici    ATAAAAATT-TATCTTAAAGTAAAAAAGCTTTAAAAACTTAGAAAGACGAGAAGACCCTA-GAACCTG-TCTATCT----------
C.murina    AATAAAATT-CCTAAAAAAGTAAAAAAGCTTTTAGGACCCAAAAAGACGACAAGACCCTA-GAACCTTCATTT----------
S.zeteki    AAACGAATTTGATCTAAAAATAAAAAAGTTTTTAATATTTAGCAAGACGATAAGACCCTA-GAACCTACTT----------
Ar.album    TATTAAATT-AATATTAAAGTAAAAAAGCTCTTTTAATTTAAAAAGACGACAAGACCCTA-AAACCTT-TTATAA----------
A.nasatu    AATTAAATT-AATTAAAGAATAAAAAAGTTCTTATTTTATAGAAAGACGATAAGACCCTA-AAACCTT-ATATGT----------
A.vul371    GGTTAAATT-AATTTAAGAATAAAAAAGTTCTTATAAACTAGAAAGACGATAAGACCCTA-AAACCTT-ATATAT----------
A.vul403    TGTTAAATT-AATCTAAGAATAAAAAAGTTCTTATAAATTAGAAAGACGATAAGACCCTA-AAACCTT-ATATAT----------
E.purpur    CATTAAATT-TATTTAAAAATAAAAAAGTTTTTATAAACTAAAAAGACGATAAGACCCTA-AAACCTTTAGT----------
C.convex    ATTTAAATTAATATAGAAAATAAAAAAGTTTTCATAAAATAGAAAGACGACAAGACCCTA-AAACCTTTATCAGGG----------
C.elonga    TCAGAAATT-TTTACAAGAATAAAAAAATTCTTATTTTCTAGAAAGACGACAAGACCCTA-AAACCTTTATTC----------
H.mengei    TTTTTAACTTTATTTAAAGGTAAAAAAGCCTTTGAATTATAAAAAGACGACAAGACCCTA-GAATCTACTATATTGCTCTAAAATACCTC
T.pusill    AATAAAATTTATTATTAAGGTTAAAAACCCTTAACTACTCAGGGGGACGACAAGACCCTA-AAATCTACTTTTTATTAAACTTTATTAGA
O.flavus    CCATTAACTTTATCAAAAGGTAAAAAAGCCTTTATTACCTAAAAGGACGACAAGACCCTA-GGATCTAAATTTTAATCTAGCTAAAAAGT
O.asellu    CTAGAAATT-AATGTAAGAATTAAAAAGTTCTTTCCCCTTAGAAAGACGACAAGACCCTA-GAACCTTCACTAGTGT----------
P.cingen    ATTGAAATT-AATATATAAATCAAAAAGTTTATTTAAATTAGATAGACGATAAGACCCTA-AAACCTTTATTGGTTATT----------
P.diladi    ATCTAAATT-AATATATAAATAAAAAAGTTTATATTCTTTAGATAGACGATAAGACCCTA-AAACCTTTATTGATAAACC----------
P.dilape    ATTTAAATT-GATATATAAATAAAAAAGTTTATATCCTTTAGATAGACGATAAGACCCTA-AAACCTTTATAGATAAACC----------
P.dispar    ACGGAAATT-AATATGTAAATAAAAAAGTTTACATCTCCTAGACGGACGATAAGACCCTA-AAACCTTTATTGATTAAAT----------
P.hoffma    ACTTAAATT-TATATAAAAATAAAAAAGTTTTTAAGTACCAGGAGGACGACAAGACCCTA-GAACCTTTACTAA----------
P.pruino    TTAAAAATT-AATATGTAGATAAAAAAGTCCACAAACCATAGATAGACGACAAGACCCTA-AAACCTTTATGGA----------
P.pruinR    ACTGAAATT-AATTAGTAGGTAAAAACCCCTATAGAATTTAGATGGACGACAAGACCCTA-AAACCTTCATCAATCC----------
P.scaber    AATGAAATT-GATAAATAAATAAAAAAGTTTATTTTAATTAGAAAGACGATAAGACCCTA-AAACCTTTACTACTTA----------
A.bipleu    TTTAAAACT-AATCTTAGGGTAAAAAAGCCTTAATTATCTAGAGGGACGATAAGACCCTG-GAACCTAATCCTTAAATTAGTACAAACCA
```


181 190 200 210 220 230 240 250 260 270
D.melano TA-TAGATTAATTAAATT--TTAATAAATAAAAATATTTTATTGGGGTGATATTAAAAT-TTAAAAAACTTTTAATTTTTTAAA----AA
D.yakuba TA-AAGATTAATTTAATT--TTAATAAATTAAAATATTTTATTGGGGTGATATTAAAAT-TTAAAAAACTTTTAATTTTAAAA-----AA
A.fransc CGGTAGGTAATTAGAC------AGAGTAAAGCAATGTTCGGTTGGGGCGACGGTAAGAACAGAATAAACACTTACAACA--TAAA---CA
A.salina CGGTAGGTAATTAGAC------AGAGTAAAACAATGTTCGGTTGGGGCGACGGTAAGAACAGAATAAACACTTACAACA--TAAA---CA
P.japoni AAATTGTTAGTATAACTTGATTTTAACGGGGGTTTGTTTCGTTGGGGCGACGGGAATATAATAAATAACTGTTCTTTTAAA-TATAATTA
P.monoFi AAATTGTTAGTATAACTTGATTTTAATTAATGTTTGTTGCGTTGGGGCGACGGGAATATAATTAGTAACTGTTCTTAAATATTTTATTAA
Cu.fagei AA----------TTAGTAAAAGAAGTTTAGCCTGGGACGGCTTTTT----AGAATAAACTAATAGAAATAA-------AA
G.locust GA----------TAGAAATTACGGTATTTGACTGGGGCGGTATGAA----TATTTAAACTATTTTTAG----------
G.zaddac TGA----------TGGTATTAAATTTATTTTACTGGGGCGGTGTTAG----TACTTAAACTAT----------TA
C.hirsut CCAAGAGATATAATC-------AATCTTATCTTTAATTATGCTGGGGCGGCAAAT-----CTTATTATCACAGCTCACCCCA------AA
C.carina ----------TTTTACTGGGGTGGTGATT-----TGATTTATCGTTTTAAAAAAAT------AA
S.serrat TCAAACTGA----------ATCAAACTATAGGGTTGTACTGGGGCGGTAAATTTT--AATCCAAACAAAACCTACTCC-------AA
S.tessie AAATCT----------AACGAAAATCAGGGTTATACTGGGGCGATAATTTT---ACTATAAACAAAAATCATCTT-------AA
E.affini ATAAGAGTTTATT---------TAAAGGCTAATCAATTTAGCTGGGGCGGCTAACTT---ATTTTTAACAAGGGTTAAAAA-------AA
A.latrei ----------AAAGGGCCTAGGAAGTTTGCTGGGGCGGCAAGTAT---GATTTTTACAGCCATATTTTT-------AA
A.aquati TGAACTAG----------TAGCAGGTTGTTGGTTAAGCTGGGGCGGCAAA------TATATAAACAATATTAATATTA------AA
J.albifr GAAGATTTTAA----------TAAGGATATACGAATTGAACTGGGGCGGTAAAT-----CGAAATGTCTGGATTAATTAT-------AA
Janirasp AA----------AAAAAGAAAAAGAATTTAGCTGGGGCGGCAAT------TAATAACACATTAATAAAGAT-------AA
I.chelip TAAGAGATCCAA----------CTATTTACCCGAGGTTTTGCTGGGGCGGCAAAT-----TTTTAATCCTACCATAATTT--------AA
I.schmit TAAGTGAAACC----------TTTTGAATAAGAGATTCTGCTGGGGCGGCAAAC-----TTTTAAACCTAACACAATTTA-------AA
I.woesne TAAGTGAAAACC----------TCTTGACTAAGGGATTCTGCTGGGGCGGCAAAC-----TTTTAAACCTAACATAATTT--------AA
H.brevic TATAGATTTTTTTTGTTG----TAATAATTATAATATTTTATTGGGGTGATGTTAAAAT-TTAATAAACTTTTAATTTTAAAG-----GT
T.europa TAAAGGAAAGAA----------CGAGTATTTATAAATTTTGCTGGGGCGGCAAG------CAGTCAAACTCTACTGATTGT-------AA
L.hypnor AACTTC----------AAACTTAAACAGGATTTTACTGGGGCGGTA--------AATAATAACAAAATTAAATTT-------AA
L.oceani AATTCACCTAAAG---------GTAACTGTTAATAATTTAGCTGGGGCGGCTG-------TAATTAAACTTTCACAAATAATA-----TA
A.offici ----------GCATAG-TTTGTTGGGGTGACATTA-----TTTTAAAACAAATAACTATTTT------AA
C.murina ----------AAATTTTGTTGGGGTGACATAGA----CTTACTTACTATTAATTC----------AA
S.zeteki ----------CTATTTACTGGGGTGGTAAT------ATTAAGTACAAGTACCATATA-------AT
Ar.album ----------TAATA-TTTACTGGGGCGGTAATTT----ACTATTGACATAAATTTAAAT-------AA
A.nasatu ----------GTATTATA-TTTACTGGGGCGGTAG-------TTTTTAATCCAAAAATTTCACT------AA
A.vul371 ----------AAATA-TTTACTGGGGCGGTAA-------TTTATTATCCATAAGTTAAAAT------AA
A.vul403 ----------AAATATTTTACTGGGGTGGTAA-------TTTATTCTCCATAAGTTAAAAT------AA
E.purpur ----------AAATATTTTACTGGGGCGGTTAATTT---ACTACTAGCATAAATTAACAA-------AA
C.convex ----------AATAATATTAACTTTTTTACTGGGGAGGTAAATAA---ATTATTAACTTAATTC----------TA
C.elonga ----------TATAAAAGAATTTTACTGGGGCAGTAATTT----ATAATTTTTTTTTAAAAACCT-------AA
H.mengei TAATACTTTA----------AGTAGCGTACGTTAG-TTTACTGGGGCGGTATAA-----ATTTTTTTCAATTTAAAAATC-------TA
T.pusill GAA----------AAATTACTTAAAAG-ATTACTGGGGCGGTAAAA-----TATCCAAACATTTTTAATAT--------TA
O.flavus TAAATCTCCT----------GTAATAGAATAAAATTATACTGGGGCGGTAATG-----ATTTTTAACAATCTTAAATAC-------TA
O.asellu ----------TAGTTTTGCTGGGGCGGTAAA------TAGTATAACGCGAGAGTTAAA-------AA
P.cingen ----------AATCAATTAACCTTTTTACTGGGGCGGTAAAT-----TATATTAACATAATTATTATC-------AA
P.diladi ----------TTTGAATATCTTTTTACTGGGGCGGAAAT------TATATTAACATAATTTAATCT--------AA
P.dilape ----------TTTAAACGTATCTTTTTACTGGGGCGGTAAAT-----TATATTAGCATAACTTAGATC-------AA
P.dispar ----------TCATATAATTATCTTTTTACTGGGGCGGTAAGT-----TATACTAACATAACTTAATAT-------AA
P.hoffma ----------TTAATGCTTGGTTTAACTGGGGCGGTTT-------ATAGATCACCTCTAGCTTCT--------TA
P.pruino ----------TCACCAACGACCTTTTTACTGGGGCGGTAG-------ATAACCCACCATAATCTATAAA------AA
P.pruinR ----------ATTATTAAGATTATTTTACTGGGGCGGTAGAT-----AATTTAAACATTATTCTATAT-------AA
P.scaber ----------AACTTTATAACGTTTTTTACTGGGGCGGTAGAT-----CATATTAGCATAATCTTATTG-------AA
A.bipleu TCAGAAACGT----------GTGCACGTGAGGGAATTTACTGGGGCGGTATAT---------TTTTCAAAATAGTTTTA-------AA

```
            271       280       290       300       310       320       330       340       350       360
D.melano    CATAAATTTA---TGAATATTTGATCCATT--AATAATGATTAAAAAATTAAGTTACTTTAGGGATAACAGCGTAATTTT-TTTGGAGAG
D.yakuba    CATTAATTTA---TGAATAATTGATCCATT--AATAATGATTAAAAAATTAAGTTACTTTAGGGATAACAGCGTAATTTT-TTTGGAGAG
A.fransc    CA--TCAATAAA-TGACC-ATTGATCC-----TTAGATGAACAAAGAC-CAAGTTACCTTAGGGATAACAGCGTAATTCTTTTTTGAGAG
A.salina    CA--TCAATAAA-TGACC-ATTGATCC-----TTAGATGAATAAAGAC-CAAGTTACCTTAGGGATAACAGCGTAATTCT-TTTTGAGAG
P.japoni    CAAAAATGTTTGGTAAATAATTGATCCTCT--ATTAGAGATTAAAAGATTAAGTTACTTTAGGGATAACAGCGTAATCTT-CTTTGAGAG
P.monoFi    CAAGTATAATTGAAGAATAATTGATCCTTT--ATTAAAGATTAAAAGATTAAGTTACTTTAGGGATAACAGCGTAATCTT-CTTTGAGAG
Cu.fagei    CAAATATCT----TGAAATTAATGACC-----CCTGAGGCCCATCTGATCAAATTACTTTAGGGATAACAGCGTAATTTA-GTTTAAAGA
G.locust    ----------TGTAA-TTTGATCCTTTA-GATAAAGAAAAATTGGTTAAGTTACTCTAGGGGTAACAGTGCTATAAT-TTTGGAGAG
G.zaddac    CTTAA--------AGAAT---AGATCCTTTA-AATAAAGAAAAAATGGAAAAGTTACTCTAGGGATAACAGTGCAATATT-TTTGGAGAG
C.hirsut    CATTTTGCTT---TGAATCAATCAACC-----TTATTCAAAAACTAATAAAAGTTACTTTAGGGATAACAGCGTAATTCT-TCTTTAGAG
C.carina    CTTAAACAA----------------------AGATTTTTTTTAAAATTTAAGCTACTCTAGGGATAACAGCGCAAAAAT-ATAAATTAG
S.serrat    CGTCTTCAA----CGAACTTCACGAACAC---TAATCAGTAAAGAAGAACAAGTTACTTTAGGGATAACAGCGCAATATC-CTTTTAGAG
S.tessie    CGTATTTAA----CGAGATTAACGAACAC---TAGTCAGTAATAAAGAACAAGTTACTTTAGGGATAACAGCGCAATATC-TTCTTAGAG
E.affini    CACCC--------TGAGTGAATTTTCTGAA--CGCGAAAGCAAGTAGATTAAGTTACTTTAGGGATAACAGCACAATCTT-TCCAGAAAG
A.latrei    TAGACTTG-----TGAGCAGTTT---------CTCCCTTTTAAGGGATTAGAGTTACTTTAGGGATAACAGCACAATTCT-CTATAATAG
A.aquati    TATTTAAAA----TAGAAAATTAGAACT----TAAGTTAAAAATTTAAACAAGTTACTTTAGGGATAACAGCATAATTTT-TCTTGATAG
J.albifr    CACGGTTTG----TGAAAATTTAGGTC-----TATTTAGATTATAGAA-AAAATTACTCCAGGGATAACAGCACAATTTT-TTTAGCAGA
Janirasp    CTTTTCAGATT--TGAAA-----GAG------CTTATACACAAATAGAATAAGTTACTTCAGGGATAACAGCACAATATT-TACATTGAG
I.chelip    CTTAACTAGCTCAAGATTTTAACGAGCC----TTCTTTAAGAAAAGGAACAAGTTACTTTAGGGATAACAGCGTTATATC-TTTTTAGAG
I.schmit    C-TAACTGTCCAAAGATTTTTACGTGCC----TTTGTAAAGAACAAGAACAAGTTACTTTAGGGATAACAGCGTTATACC-TTTTCAGAG
I.woesne    CCTAACTATCCAAAGAATTTTACGTGCC----TTTATAAAGAACAAGAACAAGTTACTTTAGGGATAACAGCGTTATACC-TTTTTAGAG
H.brevic    CATTAATTTA---TGAATAATTGATCCATT--ATTAGTGATTAAAAAAATAAGTTACTTTAGGGATAACAGCGTAATTTT-TTTGGAGAG
T.europa    CACTTTGAG----TGGATTTGATTTAGCGTT-GTTTCAAACATAAAAAACAAGTTACTTTAGGGATAACAGCACAATTTT-CTTATAGAG
L.hypnor    CACTTTGCG----TGAACTAAGAGAACAT---GTGTGCAGACAAGAGAAATAGTTACTTTAGGGATAACAGCGCAATTTT-TCCAGAGAG
L.oceani    CTCACATATTTAATGACACTAATTAACTT---CTCTAGAAACCTAAAATAAAGTTACTTTAGGGATAACAGCGCAATATC-CCAATAGAG
A.offici    CTCAT--------TAAGAGACTTAACCTA---TCTTTTAAAAAAAAGAAAAAGGTACTCTAGGGATAACAGCATAATCTT-CTTTTAGAG
C.murina    CTTAT--------TAAAAGAAATTTTAAA---ACTCTTCGAAAAAAAA-AAAGGTACTCTAGGGATAACAGCATGATTTT-TCTTTAGAG
S.zeteki    CATAT----------------------------TTTATGTACTCAAAAATACAGGTACTCTAGGGATAACAGCATAATCCC-CCTTTAGAG
Ar.album    CATAA--------TGAATGACTTTCCCTA---ACTCTATAAAATCCAA-GCAGGTACTTTAGGGATAACAGCATAATTTC-TATTTAGAG
A.nasatu    CATTA--------TGAATGATTCTGAG-----TACTCGTATAATTCAA-AAAGGTACTTTAGGGATAACAGCATAATTTT-TATTTAGAG
A.vul371    CACAA--------TGAATGATATTATC-----TATTTAATTAACTCAA-AAAGGTACTTTAGGGATAACAGCATAATTTT-TATTTAGAG
A.vul403    CATTA--------TGAATGATATTATC-----TATTTAATTAATTCAA-AAAGGTACTTTAGGGATAACAGCATAATTTT-TATTTAGAG
E.purpur    CATAA--------TGAATGTAATTCCCTAA--CTATTATTAAACACAA-GAAGGTACTTTAGGGATAACAGCATAATTTC-TATATAGAG
C.convex    CACTACATTT---TGGATGACTTTACCTAA--CTTTTTTAAAAATCAA-AAAGGTACTTTAGGGGTAACAGCGTAATTTT-TATCAAGAG
C.elonga    CTCAT--------TGAAAGACTCTACCCACCAACAGTATGTTGAAATAAAAAGGTACTTTAGGGATAACAGCATAATTTT-CATTTAGAG
H.mengei    CGTTC--------TGAACGAGTTATATTCA--ACCTACCTGTAAACAA-TTAGATACTCTAGGGATAACAGCACAATTTA-CCTCTAGTG
T.pusill    CACAAATTGTT--TGGCCTAAGTTAAC-----ACTTTTGTTACTTAAT-TTAGATACTTTAGGGATAACAGCGTTATTTT-TCTACAGAG
O.flavus    CATTTATGTA---TGATTTTTAGTACCTT---ATATTTCTGTAAAAAT-TAAGATACCCTAGGGATAACAGCACAATACA-TCTTAAGAG
O.asellu    CACTA--------TGAGTGAGATTATCTAT--CCTTGTATTGAATAAAAAAAGGTACTTTAGGGATAACAGAATAATCCT-TGCTTAGAG
P.cingen    CAAAC--------TGTTTGAATCTCAGTA---ACTTTGTAAAAAACAA-TTAGGTACTTTAGGGATAACAGCATAATTTT-TATTTAGAG
P.diladi    CACTT--------TAAATGACACCTTATAA--CTCTTTAAAATTTCAA-AAAGGTACTTTAGGGATAACAGCATAATTTT-TATTTAGAG
P.dilape    CACTT--------TGAATGGCATCCTATAA--CTCTCAAAAATTTCAA-TAAGGTACTTTAGGGATAACAGCATAATTTT-TATTTAGAG
P.dispar    CATTT--------TAAATGATCTATTCAATAACTCTTTTAAATAACAA-AAAGGTACTTTAGGGATAACAGCATAATTAT-CATTTAGAG
P.hoffma    CACCACCT-----TGAGTGAGCCCTACTAAC-TTTTAATAATAAACAA-AAAGGTACTCTAGGGATAACAGCATAATTTT-TATTGAGAG
P.pruino    CATAA--------TGGATGGCTTTAACTAA---CTTCTATAAAAACAA-AAAGGTACTTTAGGGATAACAGCATAATTCT-TATTTAGAG
P.pruinR    CATAG--------TGAATGACCAATACTAA---CTTTATAAAATACAA-AAAGGTACTTTAGGGATAACAGCACAATTCT-TATTAAGAG
P.scaber    CATAA--------TGAATGAAACCTTGAATAACTTATCAGAATTTCAA-AAAGGTACTTTAGGGATAACAGCATAATTTT-TATTAAGAG
A.bipleu    CACAG--------TGAATGATTACCTACTAA-CCTAATAAGAAAACAT-ATAGGTACTCCAGGGATAACAGCACAATCCC-TTTTTAGAG
```

```
            361       370       380       390       400       410       420       429
D.melano    ---TTCATATCGATAAAAAAGATT-GCGA-CCTCGATGTTGGATTAAGATATAATTTTGG--GT-GTAG
D.yakuba    ---TTCATATCGATAAAAAAGATT-GCGA-CCTCGATGTTGGATTAAGATATAATTTTGG--GT-GTAG
A.fransc    ---TTCAAATCGACAAAAGAGTTT-GCGA-CCTCGATGTTGGTTCAGGGACCCTACTCG---GT-GCAG
A.salina    ---TTCAAATCGACAAAAGAGTTT-GCGAGCCTCGATGTTGGTTCAGGGACCCTACTCG---GT-GCAG
P.japoni    ---TCCACATCGACAAGAAGGTTT-GCGA-CCTCGATGTTGAATTAAGGTATCCTTATAAT-GCAGCAG
P.monoFi    ---TCCTCATCGACAAGAAGGTTT-GCGA-CCTCGATGTTGAATTAAGGTATCCTTATGTT-GCAGCAG
Cu.fagei    ---ACAAATTAAAAAACTAAGATT-GCGA-CCTCGATGTTGAATTAAGGCCCCTTATCTA--GC-GCAG
G.locust    ---ACCTTATCGATAAAATTGTTT-GCAA-CCTCGATGTTGAATTATGAATGTCTAATATGAAT-GCAT
G.zaddac    ---ATCTTATCGATAAATGTGTTT-GCAA-CCTCGATGTTGAATTAGGATATCTACATG---AA-GCAG
C.hirsut    ---TTCTTATCGCAGAAAGAGATT-TCGA-CCTCGATGTTGAATTGTTTAGCTTAAAA----GA-GCAG
C.carina    ---ACCAAATATTTTATTATTATT-GCGA-CCTCGATGTTGAATTGATTTTTTATAAT----GA-GCAT
S.serrat    ---CCCTTATCGACAAGGATGTTT-ACGA-CCTCGATGTTGAATTGAAGCCCCTACTCA---GA-GCAG
S.tessie    ---CCCTTATCGACGAAGATGATT-ACGA-CCTCGATGTTGAATTGAAACCCCTGCTTA---GA-GCAG
E.affini    ---CTCCCATTGATGGAAAGGCTT-ATGA-CCTCGATGTTGAATTGGAACCCCTCCTGGC--TG-CCCG
A.latrei    ---ACCTAATTGTCTAGGGAGTTT-GTGA-CCTCGATGTTGAATTAGGGAGTCTTTAGA---TG-GCAG
A.aquati    ---TTCTTATTGGCAGAAAAGTTT-ATGA-CCTCGATGTTGAATTGAGATGTCCTTTATG--GT-GCAG
J.albifr    ---TCTTATACACAAAAAAGACTT-GTGA-CCTCGATGTTGAATTGAAGAGTCTAGAAT---AA-GCAA
Janirasp    ---ACCAGATCAGTGTAAATGACATGTGA-CCTCGATGTTGAATTGAGTGTTCTAAGCT---AA-GCAG
I.chelip    ---CTCCTATCGCCAAAGATGTTT-GCGA-CCTCGATGTTGAATTGAAATACCTGCTTT---GG-GCAA
I.schmit    ---CCCCTATCGCCAAAGGTGTTT-ACGA-CCTCGATGTTGAATTGAAATACCTATTTT---AA-GCAG
I.woesne    ---CCCCTATCGCCAAAGGTGTTT-ACGA-CCTCGATGTTGAATTGAAGTACCTGGTTT---AA-GCAG
H.brevic    ---TTCATATTGATAAAAAAGATT-GCGA-CCTCGATGTTGGATTAAGATATAATTTTAG--GT-GTAG
T.europa    ---TCCTTATCGACAAGAAAGATT-GTGA-CCTCGATGTTGAATTGAGTTGCTTTTGCT---AA-GCAG
L.hypnor    ---TCCAAATCGGCGGAAAAGATT-ACGA-CCTCGATGTTGAATTGAAATTCCTCTTATA--GA-GCAG
L.oceani    ---TCCCTATCGACTGGGATGATT-GCGA-CCTCGATGTTGAATTG-AGAACCTTCCTG---AA-GCAG
A.offici    ---CCCTAATCGACAAGAAAGATT-TTGA-CCTCGATGTTGAATTGATATTCCTTCTTA---AA-GCAG
C.murina    ---ACCTTATCGACAGAAAAGATT-TTGA-CCTCGATGTTGAATTGTAAAACCTTATGA---GA-GCAA
S.zeteki    ---CCCCTATCGACAGAGGGGATT-GTGA-CCTCGATGTTGAATTGAAGTGTCTTATTC---CA-GCAG
Ar.album    ---TTCTTATCGACATAGAAGATT-ATGA-CCTCGATGTTGAATTAATATAACTTCTAT---AA-GCAG
A.nasatu    ---TCCTTATCGACATAAAAGATT-ATGA-CCTCGATGTTGAATTGATTTAACTCTTTG---GG-GCAG
A.vul371    ---TTCTTATCGACATAAAAGATT-ATGA-CCTCGATGTTGAATTGATTTAACTATTTT---AG-GCAG
A.vul403    ---TTCTTATCGACATAAGAGATT-ATGA-CCTCGATGTTGAATTGATTTAACTATTTT---AG-GCAG
E.purpur    ---TCCTTATCGACATAGAAGATT-ATGA-CCTCGATGTTGAATTGATATAATTTTAAT---AA-GCAG
C.convex    ---ACCTAATCGACATATAAAGATTACGA-CCTCGATGTTGAATTGATCCCACTTTACT---AA-GTTA
C.elonga    ---CCCTGATCGATCTGAAAGATT-ATGA-CCTCGATGTTGAATTGGGTTCCTGATTA----GA-GCAA
H.mengei    ---CCCTTATCGATAGATAATTT--GTGA-CCTCGATGTTGAATTAGCTCTACTTAATTT--AA-GCAG
T.pusill    ---TCCCCATTGCCAGAAAAGTTT-TCGA-CCTCGATGTTGAATTGAGTTTACTAGATT---AA-GCAA
O.flavus    ---CCCTAATCGCCAGATTATTTTTGTGA-CCTCGATGTTGAATTAAGGTCCCTCTATC---AA-GCAG
O.asellu    ---TTCTTATCGACGCAGGGGTTT-ATCA-CCTCGATGTTGAATTGAGTAAACTTTATG---GG-GCAG
P.cingen    TA-AACTTATCCACATAAGAGATT-ATGA-CCTCGATGTTGAATTGATAAAACCTTATT---AA-GCAA
P.diladi    TA-AACCTATCGACATAATAGATT-ATGA-CCTCGATGTTGAATTGATAACCACTTGCT---AA-GCAG
P.dilape    TAAAACCTATCGACATAATAGATT-ATGA-CCTCGATGTTGAATTGATAACCACTTGCT---AA-GCAG
P.dispar    AA-AACCTATCGACATAATAGATT-ATGA-CCTCGATGTTGAATTGATATTGCCTTGCT---AA-GCAG
P.hoffma    ---TTCGTATCAACATAAAAGATT-ATGA-CCTCGATGTTGAATTGGTGCATCTTTATT---AG-GCAA
P.pruino    CA-ACCCTATCGACATAAGAGCTT-ATGA-CCTCGATGTTGAATTGATAATACCCCACT---AA-GCAA
P.pruinR    TA-ACCATATCGACATAAGAGATT-ATGA-CCTCGATGTTGAATTGATATTACCTCACT---AA-GCAG
P.scaber    TA-AACCTATCGACATAAAAGATT-ATGA-CCTCGATGTTGAATTGATAATACTTTACT---AA-GCAA
A.bipleu    ---CCCAAATCGGCGAGGGGGCTT-ATGA-CCTCGATGTTGAATTAATAATTCTTTTCT---AA-GCAG
```

Bibliographie

[AF87] A Apostolico and A Fraenkel. Robust transmission of unbounded strings using fibonacci representations. *IEEE Trans. Inform. Theory*, 33(2):238–245, 1987.

[AGM+90] SF Altschul, W Gish, W Miller, EW Myers, and DJ Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[APD99] L Allison, D Powell, and TI Dix. Compression and approximate matching. *The computer Journal*, 42(1):1–10, 1999.

[Ben88] CH Bennett. *Dissipation, Information, Computational Complexity and the Definition of Organization.* David Pines, addison-wesley edition, 1988.

[Ber98] V Berry. *Méthodes et algorithmes pour reconstruire les arbres de l'Evolution.* PhD thesis, Université de Montpellier II, 1998.

[BGL+98] CH Bennett, P Gacs, M Li, PMB Vitányi, and W Zurek. Information distance. *IEEE Trans. Inform. Theory*, pages 1407–1423, 1998.

[BH96] P Berman and S Hannenhalli. Fast sorting by reversal. In *7th Annual Symposium on Combinatorial Pattern Matching*, 1996.

[BP93] V Bafna and P Pevzner. Genome rearrangement and sorting by reversals. In *34th Annual IEEE Symposium on Foudations of Computer Science*, pages 148–157, November 1993.

[BP95] V Bafna and P Pevzner. Sorting permutations by transpositions. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 614–621, 1995.

[BRJ98] D Bouchon, T Rigaud, and P Juchault. Evidence for widespread wolbachia infection in isopod crustaceans: molecular identification and host feminization. *The Royal Society*, 265:1081–1090, 1998.

[Cap97] A Caprara. Sorting by reversals is difficult. In *RECOMB'97*, pages 75–83. ACM Press, 1997.

[Chr98] D Christie. *Genome Rearrangement Problems.* PhD thesis, University of Glasgow, 1998.

[CLR94] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction à l'algorithmique.* Dunod, 1994.

[CR94] M Crochemore and W Rytter. *Text Algorithms.* Oxford University Press, 1994.

[CT91] TM Cover and JA Thomas. *Elements of Information Theory.* Wiley Interscience, 1991.

[CVG95] JM Casacuberta, S Vernhettes, and MA Grandbastien. Sequence variability within the tobacco retrotransposon tnt1 population. *EMBO Journal*, 14(11):2670–2678, 1995.

[Del94] JP Delahaye. *Information, complexité et hasard.* Hermès, 1994.

[Del97] O Delgrange. *Un algorithme rapide pour une compression modulaire optimale, application à l'analyse de séquences génétiques.* PhD thesis, Université de Mons-Hainaut, Belgique, 1997.

[DT93] P Darlu and P Tassy. *Reconstruction phylogénétique.* Masson, 1993.

[EMBS99] N El-Mabrouk, D Bryant, and D Sankoff. Reconstructing the pre-doubling genome. In *RECOMB'99*, pages 154–163. ACM Press, 1999.

[Erh98] F Erhard. Phylogenetic relationships within *Oniscidea* (crustacea, isopoda). *Isr. J. Zool.*, 44:303–309, 1998.

[Fel78] J Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Syst. Zool.*, 27:401–410, 1978.

[Fel81] J Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

[Fel84] J Felsenstein. Distance methods for inferring phylogenies: A justification. *Evolution*, 38:16–24, 1984.

[Fel85] J Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39:783–791, 1985.

[FNS96] V Ferretti, JH Nadeau, and D Sankoff. Original synteny. In *7th Annual Symposium on Combinatorial Pattern Matching*, pages 159–167, 1996.

[GJK+97] Bhaskar Das Gupta, Tao Jiang, Sampath Kannan, Ming Li, and Elisabeth Sweedyk. On the complexity and approximation of syntenic distance. In *RECOMB'97*. ACM Press, 1997.

[GT93] S Grumbach and F Tahi. Compression of dna sequences. In IEEE Computer Society Press, editor, *Data Compression Conference*, 1993.

[GT94] S Grumbach and F Tahi. New challenge for compression algorithms: genetic sequences. *Journal of Information Processing and Management*, 30(6):875–886, 1994.

[GT95] S Grumbach and F Tahi. Compression des séquences nucléotidiques. *Technique et Science Informatiques*, 14(2):217–233, 1995.

[Gus97] D Gusfield. *Algorithms on strings, trees and sequences.* Cambridge university press, 1997.

[Han95a] S Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. In *7th Annual Symposium on Combinatorial Pattern Matching*, 1995.

[Han95b] S Hannenhalli. *Transforming men into mice (a computationnal theory of genome rearrangements.* PhD thesis, The Pennsylvania State University, 1995.

[HP95] S Hannenhalli and P Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *27th Annual ACM Symposium on the Theory of Computing*, pages 178–189, 1995.

[JC69] TH Jukes and CR Cantor. Evolution of protein molecules. In HN Muro, editor, *Mammalian Protein Metabolism*, volume 3, chapter 24, pages 21–132. Academic Press, New York, 1969.

[Kim80] M Kimura. A simple method for estimating evolutionary rates base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Biology*, 16:111–120, 1980.

[Kol65] AN Kolmogorov. Three approaches for definig the concept of information quantity. *Information Transmission*, 3:3–11, 1965.

[KR95] J Kececioglu and R Ravi. Algorithms for evolutionnary distances between genomes with translocation. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 604–613, 1995.

[KS93] J Kececioglu and D Sankoff. Exact and approximation algorithms for the inversion distance between two chromosomes. In *4th Annual Symposium on Combinatorial Pattern Matching*, volume 684 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, June 1993.

[LBBK91] MY Leung, BE Blaisdell, C Burge, and S Karlin. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *Journal of Molecular Biology*, 221:1367–1378, 1991.

[Lec63] Y Lecerf. Machines de turing réversibles. *Comptes Rendus*, 257:2597–2600, 1963.

[LHYN95] D Loewenstern, H Hirsh, P Yianilos, and M Noordewier. Dna sequence classification using compression based induction. Technical Report 4, DIMACS, Rutgers University, April 1995.

[Li97] WH Li. *Molecular Evolution.* Sinauer Associates Inc., 1997.

[LMR99] HP Lenhof, B Morgenstern, and K Reinert. An exact solution for the segment-to-segment multiple sequence alignment problem. *Bioinformatics*, 15:203–210, 1999.

[LP85] DJ Lipman and WR Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[LPSS84] C Lanave, G Preparata, C Saccone, and G Serio. A new method for calculating evolutionary substitution rates. *Journal of Molecular Evolution*, 20:86–93, 1984.

[LV93] M Li and PMB Vitànyï. *Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, 1993.

[MDW96] B Morgenstern, A Dress, and T Werner. Multiple dna and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93:12098–12103, October 1996.

[MFDW98] B Morgenstern, K Frech, A Dress, and T Werner. Dialign : finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998.

[Mil93] A Milosavljević. Discovering sequence similarity by the algorithmic significance method. In *1st ISMB*, pages 284–291, 1993.

[MJ93] A Milosavljević and J Jurka. Discovering simple dna sequences by the algorithmic significance method. *Computer Applications in Biosciences (CABIOS)*, 9(4):407–411, 1993.

[MSB00] A Michel-Salzat and D Bouchon. Phylogenetic analysis of mitochondrial lsu rrna in onicids (custacea isopoda). *C.R. Acad. Sci. Paris*, 2000. à paraître.

[Mye86] EW Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[Mye91] EW Myers. An overview of sequence comparison algorithms in molecular biology. Technical Report TR 91-29, The University of Arizona, http://www.cs.arizona.edu/people/gene, 1991.

[NW70] SB Needleman and CD Wunsh. A general method applicable to the search for similarities in amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[PL88] WR Pearson and DJ Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448, April 1988.

[RD98] E Rivals and JP Delahaye. Optimal representation in average using kolmogorov complexity. *Theoretical Computer Science*, 200:261–287, 1998.

[RDDD96] E Rivals, M Dauchet, JP Delahaye, and O Delgrange. Compression and genetic sequence analysis. *Biochimie*, 78:315–322, 1996.

[Riv94] E Rivals. Compression pour l'analyse de séquences. In M Crochemore, editor, *Actes de la rencontre pour les recherches de motifs dans les séquences*, pages 69–81. Institut Gaspard Monge, Université de Marne-la-Vallée, France, 1994.

[Riv96] E Rivals. *Algorithmes de compression et applications à l'analyse de séquences génétiques.* PhD thesis, Université de Lille 1, France, 1996.

[ROMM90] F Rodríguez, JL Oliver, A Marín, and JP Medina. The general stochastic model of nucleotide substitution. *Journal of Theoritical Biology*, 142:485–501, 1990.

[Sag96] MF Sagot. *Ressemblance lexicale et sctructurale entre macromolécules, formalisation et approches combinatoire.* PhD thesis, Université de Marne-la-Vallée, France, 1996.

[SD36] AH Sturtevant and T Dobzhansky. Inversions in the third chromosome of wild races of *drosophila pseudoobscura*, and their use in the study of the history of species. *Proc. Natl. Acad. Sci. USA*, 22:448–450, 1936.

[SLA+92] D Sankoff, G Leduc, N Antoine, B Paquin, BP Lang, and R Cedergen. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, 89:6575–6579, 1992.

[SM97] J Setubal and J Meidanis. *Introduction to computationnal molecular biology.* PWS Publishing Company, 1997.

[SN87] N Saitou and M Nei. The neighbor-joining method : a new method for reconstruction phylogenetic trees. *Mol. Biol. Evol.*, 4(4):406–425, 1987.

[SOWH96] DL Swofford, GJ Olsen, PJ Wadell, and DM Hillis. *Molecular Systematics (2nd edition)*, chapter Phylogenetic Inference. Snderland, USA, 1996.

[Sto88] JA Storer. *Data Compression: Methods and Theory.* Rockville, MD, Computer Sciences Press, 1988.

[SW49] CE Shannon and W Weaver. *The Mathematical Theory of Communication.* Univ. of Illinois Press, 1949.

[SW81] TF Smith and MS Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[Tah97] F Tahi. *Méthodes formelles d'analyse des séquences de nucléotides.* PhD thesis, Université de Paris-Sud, 1997.

[THG94] JD Thompson, DG Higgins, and TJ Gibson. Clustalw: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[Van43] A Vandel. Essai sur l'origine, l'évolution et la classification des *Oniscidea* (isopodes terrestres). *Supp. Bull. Biol. Fr. et Belg.*, XXX:1–136, 1943.

[Van65] A Vandel. Sur l'existence d'oniscoïdes très primitifs menant une vie aquatique et sur le polyphylétisme des isopodes terrestres. *Ann. Spéléo.*, XX:489–518, 1965.

[Var96] JS Varré. Phylogénie et compression de données. Technical report, Laboratoire d'Informatique Fondamentale de Lille, 1996.

[VDR99] JS Varré, JP Delahaye, and E Rivals. Transformation distances: A family of dissimilarity measures based on movements of segments. *Bioinformatics*, 1999.

[Wat95] MS Waterman. *Introduction to computational biology - Maps, sequences and genomes.* Chapman & Hall, 1995.

[WE87] MS Waterman and M Eggert. A new algorithm for best subsequence alignments with application to trna-rrna comparisons. *Journal of Molecular Biology*, 197:723–738, 1987.

[WS97] KH Wolf and DC Shields. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature*, 387:708–713, 1997.

[ZL77] J Ziv and A Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory.*, 23(3):337–343, 1977.

[ZL78] J Ziv and A Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory.*, 24(5):530–536, 1978.