République Algérienne Démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Amar Telidji Laghouat
Département d'Informatique

Apprentissage automatique par les algorithmes bio-inspirés : application à l'alignement multiple de séquences biologiques

Mémoire présenté pour l'obtention de grade de magister en informatique
Spécialité INGENIERIE DES SYSTEMES INFORMATIQUES (I.S.I)

Par

Rached YAGOUBI

Devant le jury composé de :

Dr. Mohamed Bachir YAGOUBI	M.C.A à l'Université Amar Telidji - Laghouat	Président
Dr. Abdelouahab MOUSSAOUI	M.C.A à l'Université Ferhat Abbas - Sétif	Rapporteur
Pr. Djelloul ZIADI	Professeur à l'Université de Rouen - France	Examinateur
Dr. Hadda CHERROUN	M.C.A à l'Université Amar Telidji - Laghouat	Examinateur

2011 - 2012

Dédicaces

A mes parents,

A mon frère et ma sœur,

A ma nièce AYA,

A toute ma famille, mes amis et mes collègues.

Remerciements

Je tiens tout d'abord à remercier vivement Dr. Abdelouahab MOUSSAOUI d'avoir proposé et encadré ce sujet. Pour le suivi régulier qu'il a effectué tout au long de ce travail ainsi que pour les conseils qu'il a su me donner.

Je souhaite également remercier Dr. Mohamed Bachir YAGOUBI, Pr. Djelloul ZIADI et Dr. Hadda CHERROUN d'avoir accepté de faire partie de mon jury.

Merci également à tous les enseignants qui ont participé à notre formation pendant l'année théorique et à tout le personnel de l'université de Laghouat.

ABSTRACT

Multiple sequence alignment is a very important task in bioinformatics, as it allows discovering and quantifying the similarities between biological molecules such as DNA, RNA, and proteins simply by comparing their sequences. It is also used as a starting point for other bioinformatic problems such as phylogeny, pattern matching and structure prediction. Although a large number of algorithms for computing multiple sequence alignment have been designed, the efficient computation of highly accurate multiple alignments is still a challenge. In this work we present *MMGA*, a hybrid bio-inspired approach for multiple sequence alignment. The design of *MMGA* is based on a combination of three algorithms : *MUSCLE*, *MAFFT* and a *genetic algorithm*. The initial population of the genetic algorithm is generated by *MUSCLE* and *MAFFT*, followed by the application of different genetic operators in order to improve the accuracy of alignments. Assessed using the popular benchmark *BAliBASE* (version 3.0), *MMGA* achieves statistically significant accuracy improvements over the existing top performing aligners, including *MUSCLE*, *MAFFT*, *ClustalW* and *ProbCons* while keeping a reduced computation time.

Keywords : Multiple sequence alignment, Bioinformatics, MMGA, MUSCLE, MAFFT, Genetic algorithm, BAliBASE.

RÉSUMÉ

L'alignement multiple de séquences est une des plus importantes taches de la bioinformatique, car il permet de découvrir et de déterminer les similitudes entre les molécules biologiques tel que l'ADN, l'ARN et les protéines par la simple comparaison de leurs séquences. Il est également utilisé comme point de départ pour d'autres problèmes de bioinformatique tel que la Phylogénie, la recherche de motif et la prédiction des structures. Bien qu'un nombre important d'algorithmes traitant le problème d'alignement multiple de séquences ont été développés, le calcul efficace des alignements multiples de grande précision reste jusqu'à présent un défi. Dans ce travail nous présentons *MMGA* (Muscle Mafft Genetic Algorithm) une approche hybride bioinspirée pour l'alignement multiple de séquences. *MMGA* combine trois algorithmes qui sont *MUSCLE*, *MAFFT* et un *algorithme génétique*. La population initiale de ce dernier est générée par *MUSCLE* et *MAFFT*, après cela nous appliquons différents opérateurs génétiques afin d'accroître la précision des alignements. L'évaluation effectuée en utilisant le benchmark populaire *BAliBASE* (version 3.0) prouve que *MMGA* réalise une amélioration importante de la précision par rapport à d'autres algorithmes performants d'alignement multiple y compris *MUSCLE*, *MAFFT*, *ClustalW* et *ProbCons* tout en maintenant un temps de calcul réduit.

Mots-clés : Alignement Multiple de séquences , Bioinformatique, MMGA, MUSCLE, MAFFT, Algorithme génétique, BAliBASE.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Afin de comprendre les mécanismes de fonctionnement du vivant, les biologistes ont besoin d'extraire des informations et des connaissances à partir des données biologiques, les interpréter et les analyser. Les données biologiques sont stockées dans des banques de données comme par exemple la banque des gènes (*GenBank*). Elle a été crée en 1982 et elle contenait 680 338 bases de nucléotides dans 606 séquences. Actuellement, dans sa version 185 datée d'aout 2011, *GenBank* contient plus de 130 milliards de bases de nucléotides dans plus de 142 millions de séquences. Avec cette vitesse de croissance des banques de données biologiques et la grande disponibilité de ces données, l'utilisation de l'outil informatique est incontournable. La *Bioinformatique* se propose comme une science capable de fournir des moyens et des outils pour satisfaire les besoins des biologistes.

La *bioinformatique* traite différents problèmes parmi lesquelles nous citons : la phylogénie, la recherche de motifs, la prédiction des structures et l'alignement de séquences. Ce dernier est très important dans la mesure où il constitue un problème à part entière, mais il est également utilisé comme point de départ pour d'autres problèmes de *bioinformatique*.

L'alignement de séquences permet de découvrir des similitudes biologiques entre les séquences (nucléiques ou protéiques) et de déterminer les correspondances entre résidus. Nous pouvons distinguer deux types de problèmes dans le domaine d'alignement de séquences selon le nombre de séquences traitées. Aligner deux séquences est appelé alignement par paires. Ce problème peut être résolu de manière exacte à l'aide de la programmation dynamique. Nous appelons alignement multiple tout alignement de plus de deux séquences. Ce problème a été démontré *NP-complet* et il ne peut être résolu par une méthode exacte que pour des séquences de petites tailles et dont le nombre est très réduit.

Différents algorithmes existent pour l'alignement multiple de séquences. Nous pouvons les classer selon trois catégories :

– Les algorithmes *exacts* sont minoritaires, ils ne peuvent être utilisés que pour des ali-

gnements ne comportant que peu de séquences car ils sont très gourmands en ressources CPU et mémoire.

- Les algorithmes *progressifs* qui consistent à aligner des sous-groupes de séquences. Ils commencent par réaliser des alignements de deux séquences, puis ces alignements sont à leur tour alignés entre eux. L'algorithme s'arrête une fois toutes les séquences regroupées. Ces méthodes sont simples, rapides et donnent généralement des alignements de bonnes qualités. Cependant, leur inconvénient majeur est la perte d'informations au cours du processus d'alignement, car traiter des séquences deux à deux est moins précis que de les traiter toutes ensemble.
- Les algorithmes *itératifs* sont basés sur des méthodes plus variées que les algorithmes progressifs. Ils ont comme point commun de réaliser l'alignement de séquences en prenant en compte toutes les séquences simultanément. Le problème de ces algorithmes est qu'ils nécessitent en générale des temps de calculs très importants.

Malgré l'existence d'un nombre très important d'algorithmes, aucun d'entre eux ne permet d'obtenir la solution optimale dans tous les cas. L'existence de *benchmark*, comme *Balibase*, permet une évaluation plus facile des nouveaux algorithmes. Elle fournit avec chaque ensemble de test la meilleure solution (un alignement référence), le résultat obtenu par un algorithme peut être évalué par une comparaison directe.

Notre mémoire est organisé en cinq chapitres. Le premier contient une introduction à la *bioinformatique*, pour cela nous présentons quelques notions de biologie moléculaire et les différents thèmes de la *bioinformatique*. Le deuxième chapitre est consacré à la présentation d'un des problèmes traités par la *bioinformatique* qui est la recherche de motifs en exposons quelques algorithmes de recherche de motifs exacts. Les deux chapitres suivants présentent en détail le problème de l'alignement de séquences. Le troisième chapitre est consacré au cas particulier de l'alignement de deux séquences. Nous y exposons le problème ainsi que des algorithmes exacts pour le résoudre. Le quatrième chapitre présente le problème d'alignement multiple de séquences proprement dit. Ce problème étant *NP-Complet*, nous exposons quelques uns des algorithmes permettant de résoudre ce problème. Le dernier chapitre contient la contribution personnelle. Nous détaillons tout d'abord l'approche proposée ensuite nous présentons les résultats obtenu et l'évaluation de l'approche en comparant avec les algorithmes existants. Le manuscrit se termine par une conclusion générale et des perspectives.

Chapitre 1

Introduction à la bioinformatique

1.1 Introduction

Depuis bien longtemps les biologistes ont eu le besoin de faire des calculs sur les données afin de pouvoir établir des nouvelles lois biologiques.

Cependant, avec le développement des ordinateurs puissants et la grande disponibilité des données biologiques (des séquences d'ADN, d'ARN ou de protéines), une nouvelle discipline a émergé de différents travaux utilisant l'outil informatique dans la recherche en biologie. Cette nouvelle discipline est connue sous la dénomination de *bioinformatique*.

Le terme bioinformatique a été introduit au début des années 1990. Le sens alors donné à ce terme correspondait à l'utilisation d'ordinateurs pour accomplir les nombreux calculs des différents problèmes d'analyse de séquences. Aujourd'hui ce sens est considérablement élargi, car nous ne considérons plus la bioinformatique comme uniquement un traitement informatique de problèmes biologiques mais plus que cela, puisqu'elle est devenue une science à part entière faisant appel aux compétences de plusieurs disciplines [1].

Dans ce chapitre nous commençons par une brève introduction à la biologie moléculaire en citant les notions biologiques de base nécessaires pour un bioinformaticien. Ensuite, nous donnerons une définition à la bioinformatique en citant quelque champs d'applications de cette dernière. Le reste du chapitre est consacré à la représentation informatique des données biologique. Nous citons, en fin de ce chapitre, quelques banques de données biologiques populaires et usuelles.

1.2 Notions de base de biologie moléculaire

Afin d'apporter des solutions informatiques efficaces aux problèmes biologiques, il faut bien avoir une certaine connaissance en biologie. C'est pourquoi nous commençons, ce chapitre, par la définition de quelques notions de base de biologie les plus utilisées en bioinformatique.

1.2.1 Cellule

Tous les êtres vivants sont constitués de *cellules*, c'est à dire d'unités structurales fondamentales limitées par une membrane (voir la figure 1.1), contenant une information génétique et capable de vivre de façon autonome quand elles sont dans un environnement favorable [2].

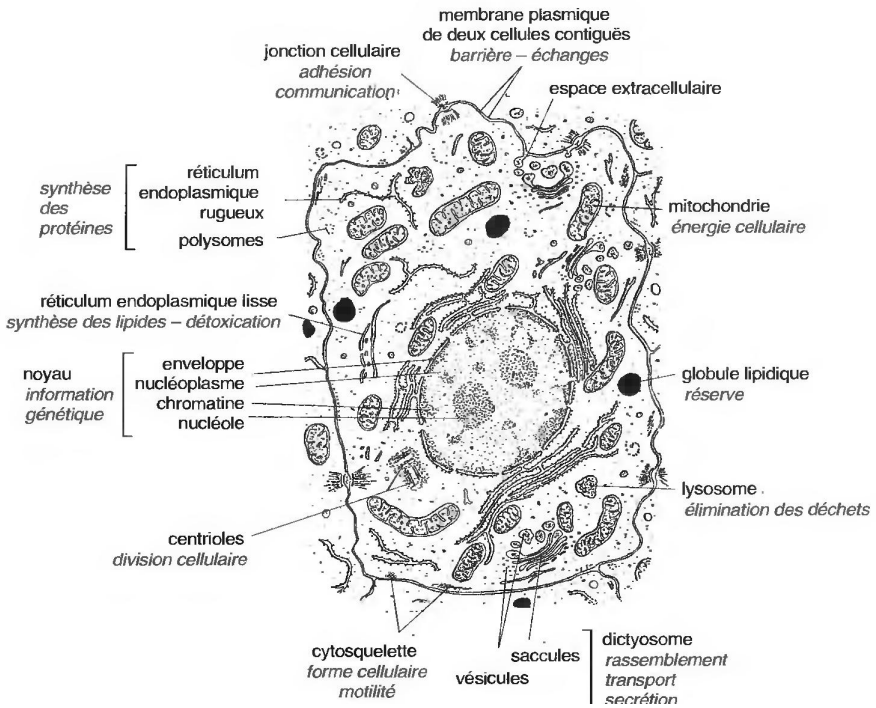

FIG. 1.1 – *Organisation d'une cellule animale* [2]

1.2.2 ADN : acide désoxyribonucléique

L'ADN est un acide de la famille des acides nucléiques, composé d'une succession de nucléotides. On peut définir l'ADN d'un organisme comme l'ensemble des informations génétiques nécessaires à l'édification, au fonctionnement et à la reproduction de chaque organisme [3].

Définition 1.1 (Nucléotides) : Les nucléotides sont constitués à partir d'une base azotée (*Adénine, Thymine, Guanine, Cytosine*) d'un sucre et d'un groupement phosphate. Par convention nous utilisons la première lettre de chacune des bases pour appeler les nucléotides [1]. Les nucléotides peuvent être rangés en deux catégories. Les bases C et T sont dites pyrimidiques, et les bases A et G sont dites puriques. Lorsque l'on parle d'ADN, la représentation que l'on fait est souvent celle d'une structure en double hélice (figure 1.2). Les deux brins constituant cette double hélice ne sont pas formés indépendamment. A chaque base d'un brin est associée sur l'autre brin une base complémentaire. Les complémentarités des bases sont les suivantes :

$$A \Longleftrightarrow T$$
$$C \Longleftrightarrow G$$

Cette complémentarité permet donc de définir totalement l'ADN à partir d'un seul des deux brins.

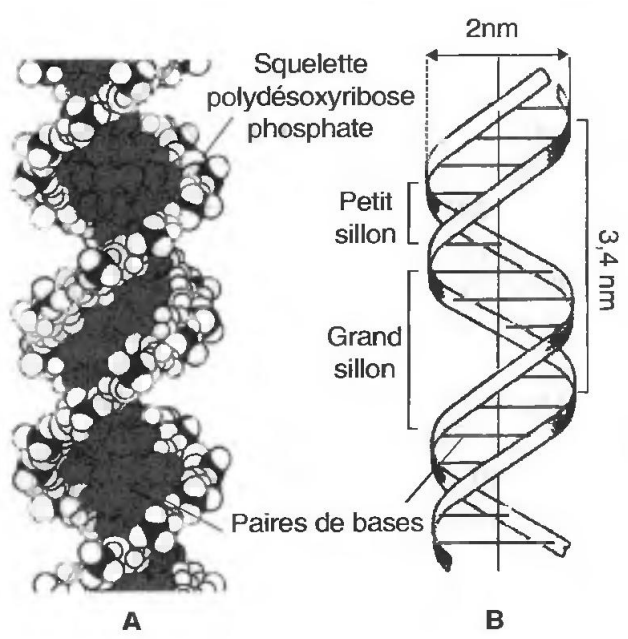

FIG. 1.2 – *La structure en double hélice d'ADN* [2]

Définition 1.2 (Séquence d'ADN) On appelle séquence d'ADN la lecture séquentielle de l'ensemble des bases constituant un brin d'ADN [1].

Définition 1.3 (Structure) Cette représentation ordonnée des bases constitue la **structure primaire** de la séquence. La représentation de l'ADN sous forme de double hélice est appelée la **structure secondaire** [1].

Définition 1.4 (Longueur) On appelle **longueur** d'une séquence d'ADN le nombre de bases qui composent cette séquence [1].

1.2.3 L'ARN : Acide Ribonucléique

L'ARN est obtenu à partir d'un des deux brins d'ADN. A chaque nucléotide de la séquence d'ADN va correspondre le nucléotide complémentaire, sauf pour l'*Adénine*. En effet, pour l'ARN, le complémentaire de l'*Adénine* n'est pas la *Thymine* mais l'*Uracile* [1].

Définition 1.5 (Transcription) La transcription correspond au transfert d'une information contenue dans la molécule d'ADN à une molécule d'ARN [2].

L'ARN est subdivisé en trois catégories, chacune ayant un rôle différent [1] :

- *L'ARN de transfert (ARNt)* est utilisé dans la phase de traduction de l'ARN en protéines.
- *L'ARN ribosomique (ARNr)* forme une trame sur les ribosomes, afin de permettre aux protéines synthétisées par les ribosomes de se fixer.
- *L'ARN messager (ARNm)* est utilisé chez les *eucaryotes* pour véhiculer l'information génétique du noyau vers le cytoplasme (substance de la cellule qui entoure le noyau).

1.2.4 Les protéines

Les protéines représentent le plus grand composant de la cellule. Elles représentent environs 50% de la matière sèche de la cellule. Elles sont responsables de presque toutes les réactions biochimiques qui ont lieu à l'intérieur de la cellule. Les protéines sont de sortes différentes et avec une variété de fonctionnalités. Certaines d'entre elles incluent [4] :

- *Protéines structurelles :* elles sont les bases de construction des divers tissus.
- *Enzymes :* elles catalysent les réactions chimiques essentielles qui auraient pris beaucoup de temps pour se produire.
- *Transporteuses :* elles portent les éléments chimiques qui font partie de l'organisme à d'autres (par exemple les hémoglobines qui portent l'oxygène).

Définition 1.6 (Protéine) Une protéine est une séquence composée d'acides aminés [1].

Dans la nature on trouve 20 différents acides aminés représentés par un alphabet de 20 lettres dont la liste est donnée dans le tableau 1.1. Deux acides aminés peuvent se joindre, avec un " lien de peptide ", formant une chaîne : un " polypeptide ".

Nom	Code	Nom	Code
Alanine	A	Leucine	L
Arginine	R	Lysine	K
Asparagine	N	Méthionine	M
Aspartate	D	Phénylalanine	F
Cystéine	C	Proline	P
Glutamate	E	Sérine	S
Glutamine	Q	Thréonine	T
Glycine	G	Tryptophane	W
Histidine	H	Tyrosine	Y
Isoleucine	I	Valine	V

TAB. 1.1 – *Liste des 20 acides aminés.*

Définition 1.7 (Traduction) La traduction est le mécanisme qui permet de synthétiser sous forme de protéine l'ARNm obtenu par transcription [1].

$$\text{ADN} \longrightarrow \text{ARNm} \longrightarrow \text{Protéines}$$

Définition 1.8 (Structure secondaire) La structure secondaire décrit le repliement local de la chaîne principale d'une protéine [1].

Définition 1.9 (Structure tertiaire) La structure tertiaire (3D), ou structure tridimensionnelle, d'une protéine correspond au repliement de la chaîne polypeptidique dans l'espace [1].

La structure secondaire ou tertiaire peut inférer la fonction d'une protéine. Donc connaître la structure va faciliter l'identification de sa fonction et par conséquent son importance pour tout l'organisme [5].

1.2.5 La génétique moléculaire

1.2.5.1 Gène et génome

Le gène est un segment d'ADN ou d'ARN. Il est situé à un endroit bien précis (locus) sur un chromosome. Chaque gène porte une information génétique et a éventuellement une

fonction précise. Sa taille varie entre des centaines et un million de bases. On distingue trois types de gènes au niveau fonctionnel : les **gènes protéiques**, les **gènes spécifiant des séquences ARN non traduits**, et les **gènes régulateurs**. Par ailleurs, le génome est l'ensemble du matériel génétique (patrimoine héréditaire) d'un individu ou d'une espèce. C'est une longue chaîne qui peut atteindre des milliards de bases [6].

1.2.5.2 Le code génétique

C'est le problème du "*dictionnaire cellulaire*" c'est-à-dire de la correspondance entre les 4 bases constituant la séquences de l'ARNm et les 20 acides aminés constituant la séquence des polypeptides.

le principe des recherches concernant le code génétique a été au départ fondé sur plusieurs hypothèses :

1. Si 1 base de l'ADN correspond à 1 acide aminé, la cellule ne pourrait coder que 4 acides aminés, ce qui est insuffisant (en effet il existe 20 acides aminés),
2. Si 2 bases correspondent à 1 acide aminé, on a 4^2 combinaisons donc la possibilité de coder 16 acide aminés (ce qui est encore insuffisant),
3. Si 3 bases correspondent à 1 acide aminé on a 4^3 combinaisons donc la possibilité de coder 64 acides aminés.

C'est donc la troisième hypothèse qui fut retenue dans la mesure où il existe 20 acides aminés. Elle fut confirmée par des mesures de tailles d'ARNm codant des protéines dont le nombre d'acides aminés était connu.

les premières expériences concluantes permettant d'établir la correspondance exacte entre 3 base (= un **codon**) et 1 acide aminé furent les expériences de *NIREMBRG* et son équipe en 1961, ils ont pu décrypté le premier codon (UUU). Après en multipliant les expériences le code génétique fut ensuite complètement décrypté (tableau 1.2) [2].

1.2.5.3 Évolution d'un Gène

Un gène peut subir des modifications et des opérations dont le résultat est souvent un nouveau gène. On peut citer quelques opérations de modifications de gènes qui peuvent survenir d'une manière spontanée ou provoquées par des acteurs externes [5] :

– *Réplication ou Duplication d'un gène :* un gène existant peut se reproduire afin de créer une paire de gènes identiques (division cellulaire).

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

TAB. 1.2 – *Le code génétique* [2]

- *Mutation :* la mutation est définie comme un changement dans la structure d'une séquence d'ADN. C'est la substitution d'un nucléotide par un autre. Ceci peut se produire lors d'une réplication. La mutation peut se manifester à une échelle plus élevée au niveau chromosomique.
- *Insertion :* elle est définie comme une insertion d'un nucléotide dans une séquence d'ADN.
- *Délétion :* c'est la disparition d'un nucléotide d'une séquence sans qu'il soit remplacé par un autre.
- *Croisement de gènes ou recombinaison :* deux gènes peuvent être cassés et puis reliés pour former un nouveau gène hybride composé des segments de l'ADN qui appartenaient aux gènes séparés.
- *Transfert (intercellulaire) horizontal :* un morceau d'ADN peut être transféré à partir du génome d'une cellule à une autre (même d'une espèce à une autre : cas des virus).

Chacune de ces modifications laisse une trace caractéristique dans la séquence d'ADN de l'organisme en affectant son génotype par conséquence son phénotype.

1.3 Définition et thèmes en bioinformatique

Définition 1.10 (bio - informatique) science qui conceptualise la biologie en termes de molécules (dans le sens de la chimie-physique) et applique des " techniques d'informatiques " pour comprendre et organiser l'information liée à ces molécules, sur une grande échelle. En bref, la bioinformatique est un système intégré de gestion pour la biologie moléculaire et a beaucoup d'applications pratiques [7] .

Le mot "bioinformatique" découle donc de l'analyse par ordinateur des données biologiques. Ces données représentent l'information stockée dans le code génétique, mais également des résultats expérimentaux de diverses sources et des statistiques,... etc.

La bioinformatique est une science récente qui évolue rapidement et qui est fortement interdisciplinaire, elle conjugue plusieurs sciences telles que la biologie moléculaire, l'informatique, et les mathématiques (statistiques)... etc. Le but de la recherche dans la bioinformatique est l'organisation et l'extraction des données, la mise en application des algorithmes complexes et le développement des outils de visualisation afin d'atteindre une compréhension exhaustive et une exploitation des informations contenues dans les séquences d'un génome.

Le rapport entre l'informatique et la biologie moléculaire est normal pour plusieurs raisons. D'abord, le taux phénoménal de données biologiques produites fournit des défis : des quantités massives de données doivent être stockées, analysées, et doivent être rendues accessibles [8] . En second lieu, les données sont souvent exprimées comme des formules statistiques, et par conséquent le calcul, est nécessaire. Ceci s'applique en particulier aux informations sur la construction des protéines et de l'organisation temporelle et spatiale de leur expression dans la cellule. Troisièmement il y a une analogie forte entre la séquence d'ADN et un programme machine[7] .

Dans ce qui suit, nous présentons quelques uns des principaux thèmes de la bioinformatique :

1.3.1 Alignement de séquences

L'alignement de séquences est une problématique importante de la bioinformatique [9] , [10] , [11] . En effet aligner des séquences constitue un problème à part entière, mais il est également utilisé comme point de départ pour d'autres problèmes de bioinformatique. Le problème de l'alignement de séquence sera détaillé un peu plus loin dans ce manuscrit.

1.3.2 Phylogénie

Les arbres phylogénétiques [12] fournissent une méthode simple pour déterminer les relations existantes entre plusieurs espèces. Pour cela le point de vue utilisé est celui de l'évolution. En effet l'objectif est de créer un arbre montrant la proximité de ces espèces. On suppose qu'à l'origine elles ont toutes un ancêtre commun. Celui-ci est représenté par la racine d'un arbre dont les feuilles représentent les espèces observées.

Construire un arbre revient à donner un scénario possible pour l'évolution depuis l'ancêtre commun jusqu'aux espèces actuelles. Le nombre d'arbres possibles augmente de façon exponentielle en fonction du nombre de feuilles. Il convient donc d'avoir un critère pertinent pour déterminer le meilleur arbre possible, c'est-à-dire correspondant au scénario le plus probable. Dans la mesure du possible, pour que cela ait un sens, il faut que les séquences aient un lien de parenté.

1.3.3 Recherche de motifs

Un motif (ou *Pattern*) au sens bioinformatique du terme [13], représente une expression qui permet de caractériser un ensemble de séquences d'ADN, d'ARN ou de protéines. Le motif peut concerner les structures primaires, secondaires ou tertiaires. Le motif trouve notamment son intérêt dans la caractérisation des fonctions des protéines : si on était capable d'exhiber un motif pour chaque fonction alors on serait en mesure de prédire automatiquement la fonction associée à une protéine.

On distingue deux notions dans le domaine de recherche de motif :

- **La découverte de motifs** qui, étant donné un ensemble de séquences, tente d'exhiber un motif commun à ces séquences. Il s'agit d'un problème complexe car on ne sait pas ce qui doit être trouvé. Dans le cas de séquences similaires, on peut utiliser un alignement multiple de séquences afin de trouver un motif simple.
- **La recherche de motif** à proprement parlé, qui concerne la détection d'un motif donné sur un ensemble de séquences. Ce problème sera détaillé dans le chapitre suivant.

1.3.4 Prédiction de structures

Le nombre de structures primaires possibles pour les protéines est exponentiel en fonction de leurs longueurs. En revanche le nombre de combinaisons de structures tridimensionnelles est beaucoup plus réduit. Ainsi, des séquences très différentes peuvent avoir des structures

similaires. La structure tridimensionnelle d'une séquence est une information contenue dans sa structure primaire. Cependant, cette information est actuellement difficile à déterminer sans utiliser une analyse directe de la séquence [14] .

Connaître la structure primaire d'une protéine ne permet pas actuellement d'en déduire sa structure tridimensionnelle.

La structure 3D d'une protéine peut être déterminée expérimentalement par *cristallographie* ou par *résonance magnétique nucléaire*. Ces méthodes sont toutefois assez lourdes à mettre en œuvre, et nécessitent un matériel spécialisé. La prédiction de structures est une branche de la bioinformatique qui consiste à essayer de déterminer la structure d'une protéine sans passer par la phase expérimentale.

La méthode qui donne les meilleurs résultats actuellement procède par homologie, c'est-à-dire en se basant sur des séquences ayant des structures primaires assez proches, et dont la structure tridimensionnelle est déjà connue. Il s'agit là d'une application directe du problème d'alignement de séquences.

1.4 Représentation informatique

Nous abordons dans cette section le point de vue informatique pour la représentation des données biologiques. Les notions vues à la section précédente peuvent être formalisées. Il devient ainsi possible de les représenter aisément pour un traitement informatique.

1.4.1 Définitions

Nous reprenons ici quelques unes des définitions énoncées précédemment. Pour cela nous les généralisons de façon à ce qu'elles puissent être utilisées aussi bien pour l'ADN, l'ARN ou les protéines.

Définition 1.11 (Alphabet) On appelle alphabet tout ensemble fini Σ de symboles distincts deux à deux.

Ainsi l'ADN et l'ARN sont représentés chacun par un ensemble de quatre lettres, et les protéines sont représentées par un ensemble de 20 lettres.

Définition 1.12 (Séquence) On appelle séquence S sur un alphabet Σ une suite ordonnée d'éléments appartenant à Σ, $S = \langle x_1, x_2, ..., x_n \rangle$

Définition 1.13 (Longueur) On appelle longueur d'une séquence le nombre d'éléments

qui la composent. On la note $|S| = n$.

Définition 1.14 (Sous-chaine) Soit S une séquence de longueur n. On appelle sous-chaine (sous-séquence) de S toute partie de S composée d'un ensemble de caractères consécutifs de S. Nous noterons $S[i..j]$ avec $1 \leq i \leq j \leq n$ la sous-séquence $S = \langle x_i, ...x_j \rangle$. Nous avons en particulier $S[i..i] = S[i] = \langle x_i \rangle$.

Définition 1.15 (Concaténation) La concaténation de deux mots u et v est le mot composé des lettres de u, suivi des lettres de v. Elle est notée uv.

Définition 1.16 (Facteur) Un mot u est un facteur du mot v, si, et seulement s'il existe deux mots w et z tels que $v = wuz$.

Définition 1.17 (Préfixe) Un préfixe u d'un mot w est un mot qui est un début de w, c'est-à-dire un mot u tel qu'il existe un mot v vérifiant $uv = w$.

Définition 1.18 (Suffixe) Un suffixe v d'un mot w est une fin de w, c'est-à-dire un mot v tel qu'il existe un mot u vérifiant $uv = w$.

Définition 1.19 (Bord) Un bord d'un mot w est un mot u, différent de w, qui est à la fois préfixe et suffixe de w.

1.4.2 Représentation informatique, format de séquence

Représenter un alphabet en informatique est une chose aisée, d'autant que les séquences qui sont manipulées peuvent être codées avec des caractères *ASCII*. Une première représentation intuitive consiste donc à considérer une séquence comme une simple chaîne de caractères.

Chaque caractère appartenant à l'alphabet sur lequel cette séquence est définie. La manipulation des chaînes de caractères est souvent assez simple en informatique, car faisant partie des types de base de tous les langages. Le stockage peut également être réalisé très simplement dans des fichiers au format texte.

Il existe de nombreux formats de séquences : plus d'une trentaine sont répertoriés et utilisés [1]. Cette multiplicité est en partie due aux informations conservées pour chacune des séquences. Un format de séquences peut généralement avoir deux provenances :

- Une base de données, où le format est utilisé pour contenir les informations conservées pour les séquences. Le format est donc spécifique aux champs d'applications de cette base de données.
- Un logiciel, où le format doit permettre d'obtenir toutes les informations nécessaires

aux calculs.

1.5 Les Banques de Données Biologiques

Les premières banques de données biologiques sont apparues au début des années 80 sous l'initiative de quelques équipes de recherches. Leur principale mission est de rendre publiques les séquences qui ont été déterminées.

Les données biologiques stockées dans ces banques sont des séquences primaires d'ADN, d'ARN et de protéines. Les données peuvent être soumises et consultées par l'intermédiaire du Web. Les séquences stockées dans ces banques sont obtenues de plusieurs manières différentes. Il y a celles isolées à partir d'une cellule, déduites à partir de la séquence nucléique par simple traduction (cas des séquences d'ARN ou protéines) ou encore par génie génétique.

Les données stockées doivent être consultées d'une manière significative, et souvent le contenu de plusieurs banques de données doit être consulté simultanément et en corrélation les uns avec les autres. Des langages spéciaux ont été développés pour faciliter cette tâche (tels que le système de récupération de séquence $\langle SRS \rangle$[1] et le système $\langle Entrez \rangle$[2]). Certaines bases de données fournissent la fonctionnalité d'accès aux séquences mais encore des liens vers d'autres bases de données et les résultats d'analyse déjà obtenus [15].

Il existe une diversité de banques, citons à titre d'exemple : GenBank, EMBL Nucleotide Sequence Database, DDBJ (DNA Data Bank of Japan) comme banque nucléique, PDB (Protein Data Bank), SWISS-PROT comme banques protéiques et PROSITE comme banque de motifs. Dans [16] sont répertoriées plus de 200 bases de données, classées par catégories de séquences.

1.6 Conclusion

La disponibilité d'une grande quantité d'informations biologiques implique toujours plus de données à traiter. Or les domaines d'étude classique de la biologie ne permettent pas de les traiter de façon efficace. Les méthodes informatiques sont devenues indispensable, ce qui a donné naissance à la bioinformatique.

La bioinformatique est une discipline récente, qui fait appel aux compétences de plusieurs

[1] Sequence Retrieval System

[2] http ://www.ncbi.nih.gov/Entrez/

disciplines scientifiques. Il y a principalement les mathématiques et l'informatique, mais également dans certains cas la physique ou la chimie. La bioinformatique regroupe donc une partie de chacun de ces domaines, ainsi que la biologie elle-même.

Dans ce chapitre, nous avons commencé par donner quelques notions de base sur la biologie moléculaires nécessaire pour un bioinformaticien. Ensuite nous avons donné une définition de la bioinformatique et quelques problèmes issus comme la recherche de motifs qui sera détaillée dans le prochain chapitre. Le problème d'alignement de séquence fera l'objet des chapitres 3 et 4. Enfin nous avons définit les techniques de représentation informatique des données biologiques et nous avons terminé le chapitre par la présentation de quelques banques de données biologiques populaires.

Chapitre 2

Les algorithmes de recherche de motifs

2.1 Introduction

La recherche de motifs dans un texte est un problème important qui apparaît dans de nombreux domaines scientifiques. En informatique, on le rencontre naturellement en traitement des données, dans l'édition de textes, en analyse syntaxique ou en recherche d'informations.

Dans sa forme la plus simple, le problème se ramène à localiser une occurrence d'un mot, le motif, dans une chaîne de caractères, le texte. Au sens bioinformatique du terme [13], un motif représente une expression qui permet de caractériser un ensemble de séquences d'ADN, d'ARN ou de protéines. Le motif peut concerner les structures primaires, secondaires ou tertiaires.

Dans ce chapitre nous commençons par donnée une classification des algorithmes de recherche de motifs, ensuite nous détaillerons quelques algorithmes importants comme celui de Knuth, Morris et Pratt et celui de Boyer et Moore.

2.2 Classification des algorithmes de recherche de motifs :

Un *motif* représente un langage non vide ne contenant pas le mot vide. Il peut être décrit par un mot, par un ensemble fini de mots, ou autrement. Le problème de la *recherche de*

motifs est celui de la localisation d'occurrences de mots du langage dans d'autres mots - ou dans des *textes* pour parler de manière moins formelle [17] .

Il existe un grand nombre d'algorithmes de recherche de motifs, ces algorithmes n'ont cessé d'évoluer depuis près de 30 ans pour limiter au maximum. Nous allons donnée une classification de ces algorithmes (voir le tableau 2.1) selon la manière de comparer le motif x.

comparer le motif x	**Algorithmes**
de doite à gauche	Naive algorithm, Morris and Pratt algorithm, Knuth Morris and Pratt algorithm, Shift-Or algorithm, Forward Dawgn Matching algorithm, Apostolico-Crochemore algorithm, Not So Naive algorithm...
de gauche à droite	Boyer-Moore algorithm, Turbo-BM algorithm, Reverse COlussi algorithm, Reverse Factor algorithm, Turbo Reverse Factor algorithm, Backward Oracle Matching algorithm, Zhu-Takaoka algorithm, Berry-Ravindran algorithm...
dans un ordre précis	Colussi algorithm, Galil-Giancarlo algorithm, Sunday's Optimal Mismatch algorithm, Maximal Shift algorithm, Skip Search algorithm, KmPSkip algorithm, Alpha Skip Search algorithm...
dans n'importe quelle ordre	Horspool algorithm, Quick Search algorithm of Sunday, Tuned Boyer-Moore of Hume and Sunday algorithm, Smith algorithm, Raita algorithm...

TAB. 2.1 – *Classification des algorithmes de recherche de motifs* [18].

Les algorithmes donnés dans le tableau 2.1 sont appelés les algorithmes de recherche de motifs exactes ou bien "*Exact pattern matching*". Il existe une autre notion appelé recherche de motifs approchés "*Approximate pattern matching*" qui est analogue au problème d'alignement de séquences [19], ce dernier sera détaillé dans le prochain chapitre.

2.3 Recherche d'un motif

Le problème que nous abordons dans cette section est le suivant : étant donné un *texte* $t \in A^*$ et un *motif* $x \in A^*$ où A est un alphabet. Nous cherchons à déterminer si x est un facteur de t, et le cas échéant trouver une occurrence de x dans t. L'efficacité d'un algorithme de recherche se mesure en nombre de comparaisons de caractères [20].

Posons $t = t_1...t_n$ et $x = x_1...x_m$, où les t_j et les x_i sont des lettres. Les algorithmes que nous présentons sont bâtis sur le schéma que voici : le motif x est comparé aux facteurs $t_{k+1}...t_{k+m}$ de longueur m de t jusqu'à trouver une coïncidence, si elle existe. L'algorithme naïf fait cette comparaison pour toutes les positions $k = 0, ..., n - m$. Les améliorations que nous examinons ensuite ne font la comparaison que pour un sous-ensemble des positions, en tirant profit de la connaissance du texte t accumulée lors des comparaisons précédentes et d'un prétraitement du motif x. Le schéma général est donc :

RECHERCHE-D'UN-MOTIF(x,t) ;
$k := 0$;
tester l'égalité $x = t_{k+1}...t_{k+m}$;
s'il y a égalité, reporter k et arrêter
sinon augmenter k et recommencer.

2.3.1 Un algorithme naïf

Posons $t_1...t_n$ et $x = x_1...x_m$. L'algorithmes naïf consiste à comparer le motif x à chaque facteur de t de longueur m. Si une occurrence est rencontrée, on la signale ; sinon, on recommence avec le facteur suivant de t. Le but est de calculer un entier k où commence une occurrence de x, c'est-à-dire tel que

$$x = x_1...x_m = t_{k+1}...t_{k+m}$$

L'algorithmes "naïf" est comme suit (Algorithme 1) :

```
procédure RECHERCHE-NAÏVE(x,t);
début
    i := 0 ; j := 1 ;
    tant que i ≤ m et j ≤ n faire
        si t[j] = x[i] alors
            i := i + 1 ; j := j + 1
        sinon
            j := j − i + 2 ; i := 1
        fin
    fin
    si i > m alors
        occurrence de x à la position j − m
    sinon
        pas d'occurrence
    fin
fin
```

Algorithme 1 : *L'algorithme Naïf*

Dans le cas le plus défavorable (voir la figure 2.1), le nombre de comparaisons est $(n - m + 1)m = nm - m^2 + m$. Pour m petit devant n, ce nombre est de l'ordre de $nm = |x||t|$ [20].

FIG. 2.1 – *Exemple d'un cas défavorable*

2.3.2 L'algorithme de Morris et Pratt

La lenteur de l'algorithme naïf, dans le cas le plus défavorable, s'explique par le fait qu'en cas d'échec, il recommence à zéro la comparaison du motif x au facteur suivant de t,

sans exploiter l'information contenue dans la réussite partielle de la tentative précédente. Si l'échec s'est produit à la i *ème* lettre du motif x, on a pour un entier k

$$x_1...x_{i-1} = t_{k+1}...t_{k+i-1} \qquad \text{et} \qquad x_i \neq t_{k+i}$$

Toute recherche ultérieure qui commence dans le facteur $t_{k+1}...t_{k+i-1}$ peut tirer profit du fait que le mot $t_{k+1}...t_{k+i-1}$ est un préfixe du motif x. Si une nouvelle recherche commence en position *l+1* (avec $k < l < k + i - 1$), elle compare $t_{l+1}t_{l+2}...$ à $x_1x_2...$. Or $t_{l+1}t_{l+2}...$ est un suffixe de $t_{k+1}...t_{k+i-1}$ qui lui est égal à $x_1...x_{i-1}$. En d'autres termes, on compare $x_1x_2...$ à un suffixe de $x_1...x_{i-1}$, donc un morceau du motif lui-même.

Ces comparaisons sont indépendantes du texte t, puisqu'elles ne concernent que des morceaux du motif. On peut donc les faire avant de considérer t, et on peut en attendre un gain de temps substantiel dans la mesure où ce prétraitement sur le motif ne sera fait qu'une seule fois et qu'il permettra d'éviter, lors de l'examen du texte, de répéter des comparaisons identiques à plusieurs endroits différents du texte.

Le prétraitement sur le motif x que nous allons réaliser permettra de reconnaître rapidement les seules configurations où la recherche d'une occurrence vaut la peine d'être continuée. Pour cela, il s'agit de déterminer les indices i où le mot $x_1...x_{i-1}$ se termine par un préfixe de x. Soit u un mot quelconque non vide ; un *bord* de u est un mot distinct de u qui est à la fois préfixe et suffixe de u.

Exemple : motif = abacabac

Motif	a	b	a	c	a	b	a	c
Bord	ε	ε	a	ε	a	ab	aba	abac

Donc il faut chercher tous les *bords* du motif x pour pouvoir décaler x d'une longueur appropriée pour le superposer sur son plus grand bord (figure 2.2).

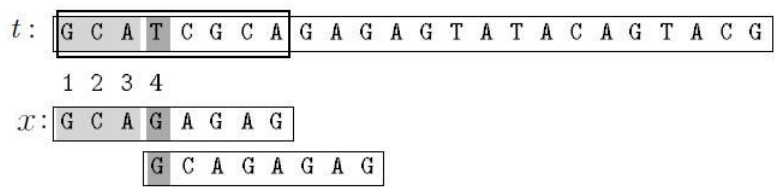

FIG. 2.2 – *Exemple de décalage de motif (Morris et Pratt)* [18]

Dans la figure 2.2 nous avons un échec à la quatrième position, le plus grand bord de la partie de déjà testé du motif ($G\ A\ C\ G$) est G. L'algorithme superpose le préfixe sur le suffixe et effectue un décalage de 3 positions à la place d'une avec l'algorithme naïf.

Dans la pratique, on utilise une fonction *s(i)* appelée la *fonction de suppléance* du motif (Algorithme 2). Cette fonction permet de déterminé le prochain décalage.

```
procédure SUPPLEANCE(x,s);
début
    s[1] := 0;
    pour j de 1 à m-1 faire
        i := s[j];
        tant que i > 0 et x[j] ≠ x[i] faire  i := s[i] ;
        s[j + 1] := i + 1
    fin
fin
```

Algorithme 2 : *Procédure de suppléance (Morris et Pratt)*

L'algorithme de MORRIS et PRATT est comme suit (Algorithme 3) :

```
procédure MORRIS-PRATT(x,t);
début
    i := 1 ; j := 1 ;
    tant que i ≤ m et j ≤ n faire
        si t[j] ≠ x[i] alors
            i := s(i)
        sinon
            i := i + 1 ; j := j + 1
        fin
    fin
     si i > m alors
        occurrence de x à la position j − m
    sinon
        pas d'occurrence de x dans t
    fin
fin
```

Algorithme 3 : *L'algorithme de MORRIS et PRATT*

L'algorithme de Morris et Pratt calcule une occurrence d'un motif x dans un texte t en au plus $2n - 1$ comparaisons de caractères, si l'on dispose de la fonction de suppléance sur x [20].

2.3.3 L'algorithme de Knuth,Morris et Pratt

L'algorithme de Knuth, Morris et Pratt [21] que nous présentons maintenant est une amélioration de l'algorithme précédent, basée sur l'élimination de situations qu'il est inutile d'examiner. Pour cela, L'algorithme exige de ne pas se retrouver dans la même situation que précédemment (voir la figure 2.3).

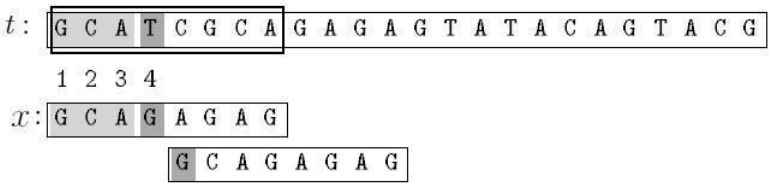

FIG. 2.3 – *Exemple de décalage de motif (Knuth,Morris et Pratt)* [18]

Dans la figure 2.3, nous remarquons qu'à la place de superposé le préfixe(G) sur le suffixe (G) et de commencé le nouveau test dans la position 4 du texte, l'algorithme de Knuth,Moriss et Pratt propose de ne pas refaire cette comparaison(le G avec le T) et de commencer directement à la prochaine position.

Cette amélioration est réalisé on introduisant une nouvelle fonction *r(i)* qui sera appelée *deuxième suppléance* (Algorithme 4).

```
procédure DEUXIEME-SUPPLEANCE(x,r);
début
    r[1] := 0 ; i := 0;
    pour j de 1 à m-1 faire
        tant que i > 0 et x[j] ≠ x[i] faire  i := r[i];
        i := i + 1;
        si x[j + 1] ≠ x[i] alors
            r[j + 1] := i
        sinon
            r[j + 1] := r[i]
        fin
    fin
fin
```

Algorithme 4 : *Procédure de deuxième suppléance (KNUTH,MORRIS et PRATT)*

l'algorithme de KNUTH,MORRIS et PRATT est comme suit (Algorithme 5) :

```
procédure KNUTH-MORRIS-PRATT(x,t);
début
    i := 1 ; j := 1 ;
    tant que i ≤ m et j ≤ n faire
        tant que i > 0 et t[j] ≠ x[i] faire i := r[i];
        i := i + 1 ; j := j + 1
    fin
    si i > m alors
        occurrence de x à la position j − m
    sinon
        pas d'occurrence de x dans t
    fin
fin
```

Algorithme 5 : *L'algorithme de KNUTH, MORRIS et PRATT*

Il est clair que l'algorithme de Knuth, Morris et Pratt est plus efficace que l'algorithme de Morris et Pratt, même si, dans le cas le plus défavorable, leur complexité est la même [20].

2.3.4 Algorithme de Horspool

L'algorithme de Horspool se distingue des algorithmes précédents dans la manière de comparer le motif x aux facteurs du texte. Alors que l'algorithme naïf et l'algorithme de Knuth, Morris et Pratt comparent les lettres du motif x aux lettres du facteur de t de la gauche vers la droite, l'algorithme de Horspool les comparent de la droite vers la gauche (voir la figure 2.4), tel que les lettre en gris son les seuls à être comparer.

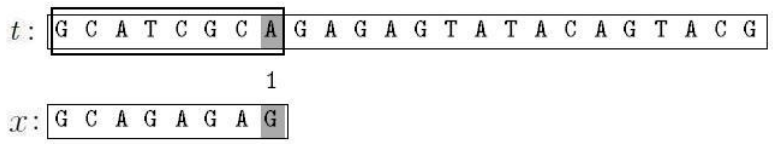

FIG. 2.4 – *La comparaison du motif de droite à gauche (Horspool)* [18]

Soit x un mot de longueur m. Pour toute lettre a de l'alphabet, soit $d(a)$ la distance entre la dernière occurrence de a dans x (la dernière place exceptée) et la dernière lettre de x. Plus précisément :

$$d(a) = \begin{cases} |u| \text{ si } au \text{ est un suffixe de } x,\ u \neq \varepsilon \text{ et } a \text{ ne figure pas dans } u, \\ |x| \text{ si } a \text{ ne figure pas dans } x. \end{cases}$$

La fonction d est la *fonction de dernière occurrence*. Par exemple, pour le mot $x = GCAGAGAG$, la fonction de dernière occurrence vaut :

	A	C	G	T
d	1	6	2	8

En remarque que la lettre T ne figure pas dans le motif, alors nous effectuons un décalage qui est égale à la longueur du motif x (voir la figure 2.5).

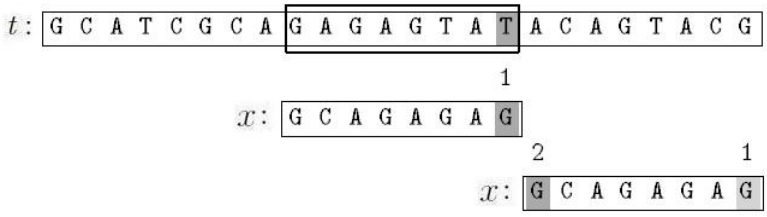

FIG. 2.5 – *Le décalage dans l'algorithme Horspool* [18]

la fonction de dernière occurrence d pour le motif x est comme suit (Algorithme 6) :

```
procédure DERNIERE-OCCURENCE(x,d);
début
    pour toute lettre a de A faire d[a] := m;
    pour i de 1 à m − 1 faire d[x_i] := m − i;
fin
```

Algorithme 6 : *La procédure de dernière occurrence (HORSPOOL)*

L'algorithme HORSPOOL est comme suit (Algorithme 7) :

```
procédure HORSPOOL(x,t);
début
    j := m;
    tant que j ≤ n faire
        i := m;
        tant que i > 0 et t_{j−m+i} = x_i faire i := i − 1;
        si i = 0 alors
            j est une fin d'occurence de x;
            j := j + 1
        sinon
            j := j + d[t_j]
        fin
    fin
fin
```

Algorithme 7 : *L'algorithme HORSPOOL*

le nombre total de comparaisons effectuées par l'algorithme Horspool est très souvent inférieur à la longueur du texte. En d'autres termes, cet algorithme n'examine pas tous les caractères, par opposition à tous les algorithmes opérant de la gauche vers la droite qui, eux, examinent au moins une fois chaque caractère. En revanche, dans le cas le plus défavorable l'algorithme a une complexité temporelle de l'ordre de $O(nm)$.

La comparaison du motif avec le texte est faite de droite à gauche, mais on pourrait aussi bien la faire de gauche à droite ; ce qui importe c'est le calcul du décalage en fonction de la dernière lettre du facteur [20].

2.3.5 L'algorithme de Boyer et Moore

L'algorithme de Boyer et Moore [22] compare les lettres du motif x aux lettres du facteur de t de la droite vers la gauche. Il fait intervenir, en plus de la fonction de dernière occurrence d, une deuxième fonction appelé *fonction du bon suffixe* qui sera notée d_2, elle prend en compte le suffixe où la différence entre le motif et le texte a été constatée. L'algorithme Horspool, défini précédemment, est une simplification de l'algorithme de Boyer et Moore.

Supposons qu'une différence est constatée entre le caractère $x_i = a$ du motif et le caractère $t_{i+j} = b$ du texte. Donc nous avons $x_{i+1}...x_m = t_{i+j+1}...t_{j+m} = u$ et $x_i \neq t_{i+j}$. le décalage effectué avec la fonction du bon suffixe consiste à aligné la partie u avec son occurrence la plus à droite dans x qui est précédée d'une lettre différente de x_i (voir la figure 2.6). Si une telle partie n'existe pas, le décalage consiste à aligné le plus long suffixe v de $t_{i+j+1}...t_{j+m}$ avec le préfixe correspondant de x (voir la figure 2.7).

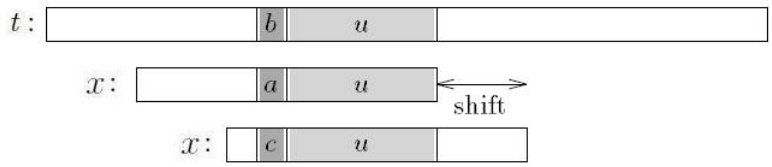

FIG. 2.6 – *Le décalage avec la fonction du bon suffixe, u réapparaît dans x et il est précédé par le caractère c différent de a.* [18]

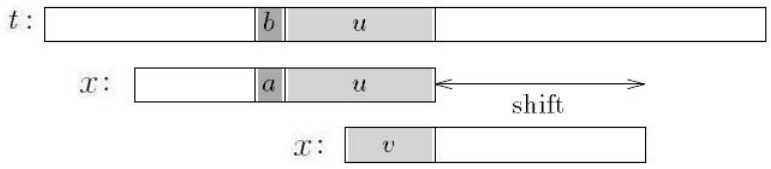

FIG. 2.7 – *Le décalage avec la fonction du bon suffixe, juste un suffixe de u réapparaît dans x.* [18]

L'algorithme de Boyer et Moore prend le maximum entre la fonction de dernière occurrence d et la fonction du bon suffixe d_2 pour déterminé un décalage (Algorithme 8) :

```
procédure BOYER-MOORE(x,t);
début
    j := m;
    tant que j ≤ n faire
        i := m;
        tant que i > 0 et t_j = x_i faire
            i := i − 1 ;j := j − 1
        fin
        si i = 0 alors
            j est une fin d'occurence de x;
            j := j + d_2[i]
        sinon
            j := j + max(d[t_j], d_2[i])
        fin
    fin
fin
```

Algorithme 8 : *L'algorithme de Boyer et Moore*

En pratique, l'algorithme de Boyer et Moore est le plus rapide des algorithmes connus. le nombre total de comparaisons est très souvent inférieur à la longueur du texte et dans sa meilleur performance l'algorithme a une complexité de $O(n/m)$. Il à été prouvé récemment qu'avec cet algorithme, le nombre de comparaisons est toujours borné par $3n$ [20].

2.4 Conclusion

le problème de recherche de motifs est très important dans la bioinformatique, et dans d'autres domaines comme par exemple dans les éditeurs de texte pour retrouver un mot donné.

Il existe de nombreux algorithmes traitant ce problème, ces algorithmes n'ont cessé d'évoluer depuis bien longtemps, en apportant souvent des petites modifications mais qui donnent des gain important en efficacité

Dans ce chapitre nous avons défini le problème de recherche de motifs, ensuite nous avons donné une classification des algorithmes de recherches de motifs exactes selon la manière dont le motif est comparé. Nous avons terminé par détailler quelques algorithmes comme

l'algorithme de Morris et Pratt ainsi qu'une amélioration de ce dernier par Knuth, Morris et Pratt, et encore l'algorithme de Boyer et Moore et ça simplification Hoorspool.

Chapitre 3

Alignement par paires

3.1 Introduction

L'alignement de séquences est une problématique importante de la bioinformatique, il constitue le point de départ pour la majorité des problèmes de recherche en bionformatique. L'alignement **par paires**, également appelé alignement de de deux séquences, est un cas particulier du problème d'alignement multiple de séquences qui sera abordé dans le chapitre suivant.

L'alignement par paires est souvent utilisé pour révéler les similarités entre les séquences, déterminer les correspondances entre résidus, localiser les motifs de conservation, étudier la régulation des gènes, et inférer les relations évolutionnaires [23]. En plus, l'alignement de deux séquences est à la base de plusieurs algorithmes d'alignement multiple de séquences.

Dans ce chapitre, nous allons présenter le problème d'alignement de deux séquences dans son ensemble. Pour cela nous allons définir quelques notions nécessaires comme la distance entre les séquences, la similarité, les matrices de substitutions et les fonctions d'évaluation. Par la suite, nous allons donner les deux stratégies d'alignements de séquences qui sont **l'alignement global** et **l'alignement local** où nous présenterons un algorithme efficace basé sur la programmation dynamique pour la résolution de chaqu'un des deux problèmes. Nous terminerons ce chapitre par la présentation de deux programmes de recherche de similarité dans les banques de séquences.

3.2 Définitions

Aligner deux séquences définies sur un alphabet consiste à couper ces séquences afin de mettre en évidence des zones communes pour faire ressortir les similarités. Ce fractionnement permet de superposer les zones identiques entre les deux séquences [1]. Pour faciliter la lecture on matérialise les coupures au sein des séquences par le symbole '−'.

Exemple : Soit les deux séquences suivantes :

- S_1 : AGAGTCACACAATTACAGGAGTAG
- S_2 : AGTATTCATACGAGGAGATTA

les séquences S_1 et S_2 sont initialement définies sur un alphabet $\Sigma = \{A, C, G, T\}$. Un alignement possible est représenté par le couple de séquences S'_1 et S'_2 définies sur un alphabet étendu $\Sigma' = \Sigma \cup \{-\}$:

S'_1 : A G − A G T C A C A C A A T T A C A G G A G T A G

S'_2 : A G T A T T C A T A C G A − − − − G G A G A T T A

Dans ce qui suit nous donnons quelques définitions nécessaires pour bien comprendre le concept d'alignement de deux séquences [1].

Définition 3.1 (Brèche) On appelle brèche ou gap dans une séquence l'insertion d'au moins un caractère '−'. La longueur d'une brèche correspond au nombre de '−' qui composent cette brèche.

Définition 3.2 (Ote lettre) soit x appartenant à l'alphabet Σ. On définit la fonction *ote* pour un élément de Σ' de la façon suivante :

$$\begin{aligned} ote : \Sigma' &\longrightarrow \Sigma \\ x &\longmapsto x \\ - &\longmapsto \phi \end{aligned}$$

Cette définition peut être appliquée à toutes les lettres composant une séquence. La définition de la fonction *ote* peut ainsi être étendue à une séquence complète.

Définition 3.3 (Ote séquences) Soit S une séquence définie sur un alphabet Σ , et soit S' une séquence définie sur $\Sigma' = \Sigma \cup \{-\}$ comme une copie de S contenant des brèches. On définit la fonction *ote* pour une séquence par :

$$\begin{aligned} ote : \Sigma'^* &\longrightarrow \Sigma^* \\ S' &\longmapsto S \end{aligned}$$

Définition 3.4 (Alignement de deux séquences) Soient $S = \langle x_1, ..., x_m \rangle$ et $T = \langle y_1, ..., y_n \rangle$ deux séquences de longueurs respectives m et n définies sur un alphabet Σ, et soient S' et T' deux séquences définies sur $\Sigma' = \Sigma \cup \{-\}$. On dit que S' et T' constituent un alignement des deux séquences S et T si, et seulement si :

- S' et T' ont la même longueur : $|S'| = |T'| = p$,
- $ote(S') = S$ et $ote(T') = T$,
- $\nexists i / S'[i] = T'[i] =$ '$-$'

Remarque : la longueur p de l'alignement de deux séquence S et T vérifie la propriété suivante :

$$\max(|S|, |T|) \leqslant p \leqslant |S| + |T|$$

Nous donnons par la suite dans le tableau 3.1 , le nombre d'alignements possible pour des séquences de longueurs assez courantes. Afin de simplifier les calculs nous supposerons que les deux séquences ont la même longueur n.

n	Nb	n	Nb
50	$1,53.10^{37}$	500	$1,53.10^{381}$
100	$2,05.10^{75}$	600	$5,01.10^{457}$
150	$3,18.10^{113}$	700	$1,66.10^{534}$
200	$5,22.10^{151}$	800	$5,59.10^{610}$
250	$8,84.10^{189}$	1000	$6,45.10^{763}$
300	$1,53.10^{228}$	1200	$7,58.10^{916}$
400	$4,76.10^{304}$	1400	$9,05.10^{1069}$

TAB. 3.1 – *Nombre d'alignements possibles de 2 séquences* [1].

D'après le tableau 3.1, nous pouvons dire que le nombre d'alignements possibles est très élevé et même pour une petite séquence de longueur n=50.

3.3 Distance et similarité entre deux séquences

3.3.1 Simarilité et homologie

Un concept important dans l'analyse de séquences est l'*homologie* des séquences [24]. Deux séquences sont dites homologues si elles possèdent une parenté du point de vue de l'évolution. Un terme relié mais différent est la *similarité* de séquences. On dit qu'il y a similarité de séquences, lorsqu'il y a de nombreuses identités entre les séquences. Pour les protéines, cela se caractérise par des paires de résidus composées de deux acides aminés appartenant à la même famille physico-chimique.

Il est important de distinguer l'homologie des séquences avec la similarité, car les deux termes sont souvent confus par quelques chercheurs qui utilisent l'un à la place de l'autre dans la littérature scientifique [1].

3.3.2 La distance de Hamming

Soient S et T deux séquences sur un alphabet Σ. On dit que deux lettres s_i et t_j de S et T se correspondent si, et seulement si, $s_i = t_j$. On utilise également le terme anglais *match* pour indiquer la correspondance de deux lettres.

Définition 3.5 (Distance de Hamming) Soient S et T deux séquences de même longueur n sur un alphabet Σ. *La distance de Hamming* [25] entre S et T, notée $d_H(S,T)$, représente le nombre de caractères $S[i]$ et $T[i]$ qui ne se correspondent pas. La distance de Hamming est définie par :

$$d_H(S,T) = |\{i \in [1..n]/S[i] \neq T[i]\}|$$

la formule précédente permet de compter le nombre de positions où les lettres ne se correspondent pas. Elle peut également s'écrire :

$$d_H(S,T) = \sum_{i=1}^{i=n} d_H(S[i],T[i])$$

Exemple : Soient S et T deux séquences tel que :

$$\begin{array}{lcccccc} S: & \text{A} & \text{G} & \text{A} & \text{C} & \text{A} & \text{T} \\ T: & \text{G} & \text{A} & \text{G} & \text{A} & \text{C} & \text{A} & \text{T} \end{array}$$

Nous remarquons que S et T n'ont pas la même longueur alors il faut comparer S à une partie de T. Donc on aura $d_H(S,T[1..6]) = 6$, par contre $d_H(S,T[2..7]) = 0$.

3.3.3 Les opérations d'édition

3.3.3.1 Définition des opérations d'édition

Les opérations d'éditions [26] définissent les différentes modifications nécessaires permettant d'expliquer l'évolution de la séquence S jusqu'à la séquence T. Ainsi, pour chaque paire de résidus, quatre cas sont possibles, correspondant chacun à une opération d'édition :

- l'*appariement* ou *match* qui correspond à deux caractères qui se correspondent : (G, G),
- la *substitution* ou *mismatch* qui correspond à deux caractères qui ne se correspondent pas : (A, C) avec $A \neq C$,
- l'ajout d'une brèche dans S ou *insertion* $(-, G)$,
- l'ajout d'une brèche dans T ou *deletion* $(A, -)$

Exemple : Soient les deux séquences S et T tel que :

S : G A C T G A G

T : G C T G G A C G

Un exemple d'alignement est donné dans la figure 3.1 contenant les différents opérations d'édition.

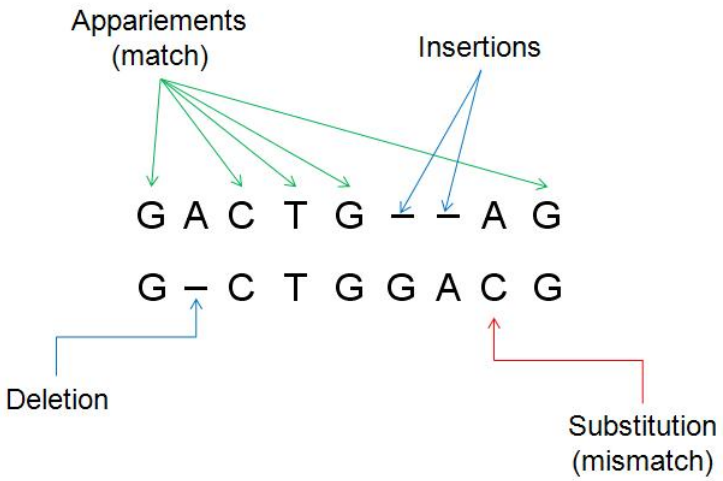

FIG. 3.1 – *Exemple d'un alignement*

3.3.3.2 La distance d'édition (distance de Levenstein)

Les opérations d'édition permettent de définir une distance entre les deux séquences appelée *distance d'édition* ou *distance de Levenstein* [27]. Pour chaque opération d'édition on associe une valeur. L'appariement correspondant à une identité a une valeur de 0, les

autres opérations ont une valeur de 1.

Définition 3.6 (Distance de Levenstein) Soient S et T deux séquences, et soit λ la somme des opérations d'édition utilisées. La distance d'édition, entre les deux séquences S et T est la valeur minimale que peut prendre λ.

3.3.4 Fonction de score

Les opérations d'édition que nous venons de définir vont permettre de transformer les séquences S et T en deux séquences S' et T' de même longueur, et contenant des brèches dans le cas des *insertions* ou *deletions*.

Le nombre d'alignements est très important, mais tous les alignements ne sont pas de qualités équivalentes. Afin de pouvoir déterminer si un alignement est meilleur qu'un autre, nous utilisons une *fonction de score*.

Définition 3.7 (Fonction de score) Soit Σ un alphabet, et soit $\Sigma' = \Sigma \cup \{-\}$. Une fonction de score pour un alphabet Σ est une application f définie par $f : \Sigma'^* \text{ x } \Sigma'^* \longrightarrow \mathbb{R}$, et qui associe la somme des valeurs de ses opérations d'éditions à un alignement [1].

Pour utiliser une fonction de score, nous avons besoin de deux éléments :

- Une matrice de substitution permettant d'associer une valeur aux opérations d'appariement et de substitution,
- Un modèle d'évaluation pour les brèches associant une valeur aux opérations d'insertion et de deletion.

Ces deux éléments serons détaillés par la suite dans ce manuscrit.

3.3.5 Les matrices de substitution

3.3.5.1 Les matrices de substitution pour l'ADN

Les Matrices de scores utilisées pour L'ADN sont relativement simple [24], une valeur positive ou un score élevé est attribué à un appariement et une valeur négative ou un score faible est attribué à une substitution. Par exemple nous pouvons utiliser la matrice d'identité, basée sur la distance d'édition, comme dans le tableau 3.2 , On peut aussi attribuer différents scores comme dans le tableau 3.3.

Les valeurs attribuées pour les appariements et les substitutions dans le tableau 3.3 sont basées sur la supposition que les fréquences de mutation sont égales pour toutes les bases.

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

TAB. 3.2 – *La matrice d'identité.*

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

TAB. 3.3 – *Exemple de matrice avec des valeurs négative.*

Cependant, en pratique les transitions (A à G, G à A, C à T, et T à C) se produisent plus fréquemment que les transversions (les autres passages entre nucléotides) [1]. Un exemple d'une matrice Transition/Transversion est donné dans le tableau 3.4.

	A	C	G	T
A	1.36	-1.6	-0.37	-1.6
C	-1.6	1.36	-1.6	-0.37
G	-0.37	-1.6	1.36	-1.6
T	-1.6	-0.37	-1.6	1.36

TAB. 3.4 – *Exemple d'une matrice Transition/Transversion.*

3.3.5.2 Les matrices de substitution pour les protéines

Pour les protéines, les matrices de substitutions ont été beaucoup plus étudiées [1]. On peut trouver principalement 2 grandes familles, mais rien n'empêche d'utiliser une autre matrice si on le souhaite.

Les matrices PAM

La famille de matrices PAM [28] ou "point accepted mutation", bien que "accepted point mutation" ou APM peut-être le terme le plus approprié, a été obtenue à partir de calculs destinés à déterminer les fréquences de remplacement d'un acide aminé par un autre au cours de l'évolution. Pour cela des groupes de séquences très similaires ont été étudiés, en cherchant à chaque fois quelles mutations étaient possibles.

Définition 3.8 (Distance PAM) Soient S_1 et S_2 deux séquences protéiques. On dit qu'elles sont à une distance de *1 PAM* si le nombre de mutations pour passer de S_1 à S_2 est de 1 pour 100 acides aminés.

D'une façon plus générale, une distance de n *PAM* signifie qu'il y a eu en moyenne n mutations pour 100 acides aminés pour l'évolution de S_1 à S_2.

Définition 3.9 (Matrices PAM) Soient S_1 et S_2 deux séquences distantes de n *PAM*. *PAM* n est la matrice qui permet de calculer le logarithme de la probabilité que S_1 évolue en S_2 .

Cette valeur prend bien sûr en compte la probabilité d'apparition de tous les acides aminés. Ainsi, pour deux acides aminés α et β, la valeur qui leur est associée dans la matrice *PAM* n indique la possibilité pour qu'il y ait une mutation de α en β dans deux séquences distantes de n *PAM*.

La première matrice à calculer est *PAM 1*. En multipliant cette matrice par elle-même, on obtient une matrice permettant d'évaluer une mutation suivie d'une seconde mutation, c'est-à-dire *PAM 2*. Ainsi, en élevant *PAM 1* à la puissance n on obtient la matrice *PAM* n.

La détermination de la distance entre deux séquences est assez difficile lorsque celles-ci ne sont pas très similaires. Il devient donc assez délicat de choisir une matrice. Le tableau 3.5 donne la correspondance entre les valeurs de *PAM* avec le taux de mutation observé des acides aminés. A titre d'exemple, la matrice *PAM 250* donnée au tableau 3.6 est une matrice souvent utilisée pour l'alignement de séquences. En pratique elle correspond à une série de mutations menant à une conservation de 20% de la séquence d'origine.

La valeur PAM	Taux de mutation observé (%)	Identité de séquence (%)
0	0	100
1	1	99
30	25	75
80	50	50
110	40	60
200	75	25
250	80	20

TAB. 3.5 – *Correspondance entre les valeurs de PAM avec le taux de mutation observé*[24].

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

TAB. 3.6 – *La matrice PAM 250*

Les matrices BLOSUM

les matrices *BLOSUM* [29] ou "Blocks Substitutions Matrices" utilisent également le principe de distance entre deux séquences. Le défaut des matrices *PAM* est de prendre des séquences trop proches les unes des autres pour les calculs. Par contre, les matrice *BLOSUM* utilisent des séquences similaires plus distantes [1].

Ces matrices utilisent pour les calculs la base de données *BLOCKS* [30]. Celle-ci contient des parties de séquences similaires alignées sous forme de blocs. Les séquences sont suffisamment proches pour que les alignements ne contiennent pas de brèches. Les différences sont donc uniquement dues à des mutations d'acides aminés.

Suivant le degré de similarité entre les blocs utilisés, il est ainsi possible de créer différentes matrices. Comme les matrices *PAM*, les matrices *BLOSUM* sont suivies d'un nombre. Celui-ci correspond au pourcentage d'identité dans les blocs utilisés pour la construction de la matrice. A titre d'exemple, la matrice *BLOSUM 62* donnée au tableau 3.7 est une matrice très utilisée, elle est obtenue avec des blocs contenant 62% d'identité d'acides aminés.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

TAB. 3.7 – *La matrice BLOSUM 62*

3.3.6 Evaluation des brèches

Accomplir un alignement optimal entre des séquences exige souvent l'utilisation des brèches (gaps) qui représentent des insertions et des deletions. L'évaluation des brèches est très importante [1]. Le nombre de brèche change selon les valeurs attribuées. Si elles ne sont pas assez pénalisantes, la fonction de score risque de favoriser les alignements qui en contiennent beaucoup. Inversement, les alignements ne comportant pas assez de brèches sont favorisés si les brèches sont trop fortement évaluées.

Il existe principalement deux modélisations [31] pour évaluer le coût engendré par l'insertion d'une brèche. Les valeurs associées à une brèche pour ces modèles peuvent être obtenues par des fonctions. Ces fonctions prennent en paramètre la longueur de la brèche et retourne le coût de celle-ci.

3.3.6.1 Les brèches à coût constant

Dans ce modèle on attribue la même valeur à tous les caractères '−'. Ainsi, si la valeur associée à une brèche de longueur 1 est α, le coût global associé à une brèche de longueur l quelconque est $\alpha.l$.

Ce modèle permet une évaluation très simple de l'alignement. Il suffit pour cela de faire la somme de toutes les valeurs attribuées aux brèches sur la longueur de l'alignement.

3.3.6.2 Les brèches à coût affine

D'un point de vue biologique, la création de la brèche est beaucoup plus pénalisante que son élongation [1]. Pour cela, il faut associer un coût plus important pour une ouverture de brèche par rapport à une extension de cette dernière.

Les dénominations généralement utilisées pour le coût d'ouverture d'une brèche sont K ou *gop* (gap opening penalty). Pour l'extension de la brèche on utilise fréquemment h ou *gep* (gap extending penalty). Le coût associé à une brèche de longueur l est alors la fonction affine $gap(l) = K + h.(l - 1)$.

Les valeurs généralement utilisées pour des matrices de substitutions standard sont de l'ordre de -10 pour K et -1 pour h. Ces valeurs peuvent bien sûr varier, mais en conservant toujours K/h voisin de 10.

Remarque : Dans le cas particulier des brèches de début et fin d'alignement, ces brèches peuvent éventuellement ne pas être comptées.

3.3.7 Les fonctions d'évaluation pour alignements par paires

Afin de déterminer la qualité d'un alignement nous avons besoin d'utiliser une fonction d'évaluation. Pour deux alignements distincts d'un même couple de séquences, l'alignement ayant la meilleure évaluation est celui qui doit être de meilleure qualité.

Pourcentage de similarité

Première méthode simple permettant d'avoir une bonne idée de la qualité de l'alignement de deux séquences [1]. Cette méthode consiste à parcourir toutes les paires de résidus de l'alignement. Lorsque les deux nucléotides (respectivement deux acides aminés dans le cas des protéines) sont identiques on attribue la valeur 1, et 0 dans les autres cas. On obtient ainsi le nombre d'identités de l'alignement.

Pour que cette valeur soit représentative, il est nécessaire de la rapporter à la longueur de l'alignement. Soit v la fonction identité, qui associe 1 à deux résidus identiques, et 0 sinon. La similarité entre deux séquences alignées S'_1 et S'_2 est donnée par :

$$Sim(A) = \sum_{i=1}^{i=l} v(S'_1[i], S'_2[i])/l$$

Cette méthode est très simple à mettre en œuvre, car elle ne fait intervenir que l'alignement lui-même. Elle est principalement utilisée pour calculer les matrices de distances entre plusieurs séquences. En effet, si l'on suppose que A est le meilleur alignement pour deux séquences S_1 et S_2, alors $1 - Sim(A)$ permet de définir une distance entre les séquences.

Utilisation d'une matrice de substitution

Le pourcentage de similarité est une méthode simple à utiliser mais elle n'est pas toujours très représentative [1]. En effet, attribuer la même valeur à tous les nucléotides n'est pas conforme à ce qui est observé dans la réalité. De plus la valeur 0 dans le cas où il n'y a pas identité est encore moins acceptable. Nous avons vu dans la partie sur les matrices de substitution que dans certains cas il n'est pas très pénalisant d'avoir une substitution de deux acides aminés.

Exemple : Dans la matrice *PAM 250* (figure 3.6)

- Dans le cas d'un appariement de l'acide aminé D la valeur attribuée est : 4
- Dans le cas d'une substitution de l'acide aminé D avec l'acide aminé E la valeur attribuée est : 3
- Dans le cas d'une substitution de l'acide aminé D avec l'acide aminé W la valeur attribuée est : -7

Autre différence notable avec le pourcentage de similarité, les brèches sont ici également prises en compte de façon plus réaliste. C'est-à-dire qu'il est nécessaire d'utiliser une évaluation plus pertinente, comme exposé précédemment.

Définition 3.10 (Evaluation d'un alignement) Soit A un alignement de longueur n des deux séquences S_1 et S_2 définies sur un alphabet Σ, et soit M une matrice de substitution pour Σ. On définit l'évaluation de A par la valeur suivante :

$$Eval(A) = \sum_{i=1}^{i=n} val(S'_1[i], S'_2[i])$$

où $val(S'_1[i], S'_2[i])$ vaut :

- $M(S'_1[i], S'_2[i])$ si $S'_1[i]$ et $S'_2[i]$ ne sont pas des brèches,
- le coût associé à une brèche sinon.

3.4 Alignement global

3.4.1 Généralités

Il existe deux différentes stratégies d'alignement qui sont souvent utilisées : *l'alignement global* et *l'alignement local* [24]. Nous commençons ici avec l'alignement global en exposant un algorithme efficace basé sur la programmation dynamique.

La programmation dynamique est un paradigme de conception qu'il est possible de voir comme une amélioration ou une adaptation de la méthode diviser et régner. Elle résout des problèmes en combinant des solutions de sous-problèmes [32].

Le problème d'alignement global de deux séquences reprend la définition d'alignement telle que nous l'avons déjà proposée. Il s'agit de trouver le meilleur alignement possible des deux séquences complètes [1].

3.4.2 Algorithme de Needleman-Wunsch

L'algorithme de Needleman-Wunsch [9] est basé sur le principe de la programmation dynamique.

Soient S et T deux séquences de longueurs p et q à aligner. Le problème $P(i, j)$ consiste à déterminer le meilleur alignement des sous-séquences $S[1..i]$ et $T[1..j]$. L'objectif pour nous est d'aligner les séquences S et T en totalité, ce qui correspond à déterminer $P(p, q)$. L'évaluation de $P(i, j)$ correspond à la valeur associée à cet alignement, nous la noterons $V(i, j)$.

Nous allons exprimer $P(i, j)$ en fonction de ses sous-problèmes. Pour déterminer $P(i, j)$, intéressons-nous à la dernière position de cet alignement. Il n'est pas possible de trouver deux brèches, donc 3 cas sont envisageables :

- $S[i]$ est aligné avec une brèche,
- $T[j]$ est aligné avec une brèche,
- $S[i]$ et $T[j]$ sont alignés.

Le premier cas correspond à $S[i]$ aligné avec une brèche, ce qui veut dire que dans cette configuration, la dernière position de l'alignement a le coût d'une brèche. Le reste de l'alignement est formé par les séquences $S[1..i-1]$ et $S[1..j]$. Or il s'agit là du problème $P(i-1, j)$, qui est un sous problème de $P(i, j)$. Ce qui veut dire que par hypothèse, $V(i-1, j)$ est connue, et donc la valeur de l'alignement avec $S[i]$ aligné avec une brèche a pour valeur

$V(i-1,j) + val(S[i],'-')$, où val est la fonction définie à la définition 3.10.

Le second cas, $T[j]$ aligné avec une brèche, est le symétrique du précédent, et il peut donc être décomposé en deux parties : $P(i,j-1)$ et l'alignement d'une brèche avec $T[j]$. La valeur associée est alors $V(i,j-1) + val('-',T[j])$.

Enfin, le dernier cas correspondant à l'alignement en dernière position de $S[i]$ avec T[j]. L'alignement du reste des deux séquences correspond au problème $P(i-1,j-1)$, qui est également un sous-problème de $P(i,j)$. La valeur associée est donc donnée par $V(i-1,j-1) + val(S[i],T[j])$.

Ces calculs effectués, il ne reste plus qu'à déterminer laquelle des trois solutions est optimale. Donc nous obtenons :

$$V(i,j) = max \begin{pmatrix} V(i-1,j) + val(S[i],'-'), \\ V(i,j-1) + val('-',T[j]), \\ V(i-1,j-1) + val(S[i],T[j]) \end{pmatrix}$$

Remarque : La formule précédente est valable dans le cas des brèche à coût constant. En revanche pour les brèches à coût affine, nous avons besoin de savoir s'il s'agit d'une création ou d'une extension de brèche lorsque l'on fait une insertion ou une deletion. Donc il faut apporter quelques modifications à la formule précédente.

Afin de déterminer $P(p,q)$ au moyen de la programmation dynamique, il est nécessaire de calculer tous ces sous-problèmes et de les conserver. On utilise pour cela une matrice de taille $[p+1, q+1]$ contenant tous les problèmes $P(i,j)$ avec $0 \leq i \leq p$ et $0 \leq j \leq q$. Traditionnellement, la représentation utilisée est celle du tableau 3.8.

	–	C	T	G	G	A
–						
C						
A						
G						
A						

TAB. 3.8 – *Exemple de tableau utilisé pour la programmation dynamique.*

Exemple : Reprenons l'exemple précédent, et cherchons l'alignement optimum avec la matrice de substitution suivante :

M	A	C	G	T
A	6	3	3	3
C	3	6	3	3
G	3	3	6	3
T	3	3	3	6

On attribue à chaque brèche un coût constant $\alpha = 1$. donc pour la première ligne et la première colonne nous avons :

- pour $i = 0$ et $j = 0 : V(0,0) = 0$ par hypothèse,
- pour $i > 0$ et $j = 0 : V(i,0) = V(i-1,0) + \alpha$,
- pour $i = 0$ et $j > 0 : V(0,j) = V(0,j-1) + \alpha$.

Le résultat de cette première étape est donné dans le tableau 3.9.

	–	C	T	G	G	A
–	0	1	2	3	4	5
C	1					
A	2					
G	3					
A	4					

TAB. 3.9 – *Calcul des valeurs associées aux problèmes de plus bas niveau.*

Voyons maintenant le calcul pour $V(1,1)$. Comme $V(0,0)$, $V(0,1)$ et V$(1,0)$ sont connues, cette valeur peut être calculée :

$$V(1,1) = max \begin{pmatrix} V(0,1) + val(C,-) \\ V(1,0) + val(-,C) \\ V(0,0) + val(C,C) \end{pmatrix}$$

$$V(1,1) = max \begin{pmatrix} 1+1 \\ 1+1 \\ 0+6 \end{pmatrix}$$

$$V(1,1) = 6$$

En continuant avec le même principe, on peut calculer successivement toutes les valeurs du tableau. On obtient les résultats donnés dans le tableau 3.10 :

	–	C	T	G	G	A
–	0	1	2	3	4	5
C	1	6	7	8	9	10
A	2	7	9	10	11	15
G	3	8	10	15	16	17
A	4	9	11	16	18	22

TAB. 3.10 – *Résultat complet.*

Construction de l'alignement

La méthode que nous venons de présenter permet de déterminer la valeur optimale $V(p, q)$ pour l'alignement des deux séquences dans le cas des brèches à coût constant. La suite des calculs effectués permettant d'obtenir cette valeur est utilisée pour construire le ou les alignements correspondants. En effet, l'optimum obtenu est la somme des valeurs des opérations d'éditions, et il est possible de retrouver chaque opération d'édition élémentaire menant à ce résultat.

Ainsi, il est possible de déterminer comment a été obtenue la valeur de $V(p, q)$. Puisque l'on sait qu'il s'agit de l'optimum obtenu à partir des trois sous-problèmes $P(p-1, q)$, $P(p, q-1)$ et $P(p-1, q-1)$. Suivant le sous-problème ayant permis de calculer $V(p, q)$, trois cas sont envisageables :

- Le cas $P(p-1, q)$ correspond à une insertion dans S,
- Le cas $P(p, q-1)$ correspond à une deletion dans T,
- Le cas $P(p-1, q-1)$ correspond à un match ou un mismatch.

La dernière position de l'alignement étant trouvée, il est possible de réitérer le même processus à partir des sous-problèmes. Il suffit en effet de trouver lequel de ces sous-problèmes a permis de l'obtenir pour avoir l'avant-dernière position de l'alignement. De proche en proche cette méthode permet de remonter jusqu'à $P(0, 0)$, et ainsi de construire l'alignement complet.

Cette méthode permet de construire un alignement de deux séquences, mais nous avons présenté la méthode en supposant qu'à chaque étape un unique sous-problème permettait d'avoir le résultat. Ce n'est bien sûr pas le cas, puisqu'il est possible que deux ou même les trois sous-problèmes conduisent au résultat optimum. Lorsque cela arrive, le processus de construction est subdivisé, et chacun permet d'obtenir un alignement différent.

D'un point de vue calculatoire, retrouver à chaque fois quels sous-problèmes ont permis d'aboutir au résultat implique de refaire les calculs. Le nombre de calculs que cela implique est négligeable lorsqu'il n'y a qu'un alignement à déterminer, mais il peut devenir important lorsqu'il y a de nombreux alignements possibles. Pour cette raison il est préférable de conserver pour chaque problème $P(i, j)$ quels sous-problèmes ont permis de l'obtenir. Il devient ainsi rapide de construire le ou les alignements des deux séquences. Dans notre exemple, en marquant à chaque étape avec une flèche les sous-problèmes utilisés, cela donne le résultat du tableau 3.11.

	_	C	T	G	G	A
_	0 →	1 →	2 →	3 →	4 →	5
C	1 ↓	6 ↘	7 →	8 →	9 →	10
A	2 ↓	7 ↓	9 ↘	10 →	11 ↘	15
G	3 ↓	8 ↓	10 ↘	15 ↓	16 →	17
A	4	9	11	16	18	22

TAB. 3.11 – *Indication à chaque étape des sous-problèmes utilisés.*

On en déduit donc les alignements suivants pour un coût de 22 :

C T G G A	C T G G A	C T G G A
C A – G A	C A G – A	C – A G A

3.5 Alignement local

3.5.1 Définition

La méthode d'alignement global que nous venons de présenter permet de déterminer le meilleur alignement des deux séquences complètes S et T. Toutefois cette optimalité sur l'ensemble des séquences correspond au meilleur compromis possible. Elle ne garantit nullement qu'il ne soit pas possible de mieux aligner deux sous-séquences de S et T [1].

Définition 3.11 (Alignement local) Soit S et T deux séquences. On définit l'alignement local de S et T comme étant l'alignement A des séquences S' et T' vérifiant :

– S' est une sous-séquence de S,

– T' est une sous-séquence de T,

– Il n'existe pas d'autres sous-séquences de S et T dont l'alignement B vérifie $Eval(B) > Eval(A)$, $Eval$ étant la fonction définie à la définition 3.10.

L'alignement local représente donc l'alignement dont l'évaluation est la meilleure parmi les alignements de toutes les sous-séquences de S et T.

3.5.2 Algorithme de Smith-Waterman

L'algorithme de *Smith et Waterman* [33] permet de déterminer l'alignement local de deux séquences. Pour cela, il est en partie basé sur l'algorithme d'alignement global basé sur le principe de la programmation dynamique.

En utilisant la définition que nous venons de donner, on constate qu'il est nécessaire de satisfaire deux conditions pour obtenir un alignement local :

– Trouver dans la matrice la valeur V maximale,

– Pour que cette valeur corresponde exactement à ce qui est attendu, aucune valeur ne doit être négative. Cette condition, permet de recommencer depuis le début un nouveau sous-alignement.

L'algorithme de *Smith et Waterman* peut être utilisé quelle que soit la méthode de coût des brèches. Dans le cas des brèches à coût constant, l'équation est :

$$V(i,j) = max \begin{pmatrix} V(i-1,j) + val(S[i],'-'), \\ V(i,j-1) + val('-',T[j]), \\ V(i-1,j-1) + val(S[i],T[j]) \\ 0 \end{pmatrix}$$

L'algorithme est alors le suivant :

1. Initialiser la première ligne et la première colonne à 0,
2. Utiliser l'algorithme de *Needleman-Wunsch* avec la nouvelle équation donnée ci dessus,
3. Rechercher la valeur maximale dans la matrice,
4. Construire l'alignement à partir de cette valeur, en arrêtant à la première valeur nulle rencontrée.

Remarque :

– La valeur 0 d'arrêt peut éventuellement être celle de coordonnées (0, 0) dans la matrice,

– L'ajout de la valeur 0 dans l'équation précédente permet de définir le début de l'alignement puisque l'on sait que la valeur suivante dans l'alignement est forcément positive. Ainsi la valeur maximale de la matrice correspond bien à la valeur réelle de l'alignement local.

Exemple d'alignement local

Dans cet exemple nous alignons deux séquences d'ADN plus longues que dans les exemples précédents. Nous pouvons ainsi voir que l'alignement local est différent de l'alignement global, et ne prend en compte qu'une partie des deux séquences. Les séquences utilisées sont :

– GCAGAGCACT

– GCTGGAAGGCAT

La matrice de substitution utilisée est la suivante :

M	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

Les brèche sont à coût constant : -7.

Le détail des calculs ainsi que le chemin suivi pour la construction sont donnés dans le tableau 3.12. La valeur maximale calculée est 19. Elle constitue donc le point de départ pour la construction de l'alignement local.

	-	G	C	T	G	G	A	A	G	G	C	A	T
-	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	0	0	5	**5**	0	0	5	5	0	0	0
C	0	0	10	3	0	1	**1**	0	0	1	10	3	0
A	0	0	3	6	0	0	6	**6**	0	0	3	15	8
G	0	5	0	0	11	5	0	2	**11**	5	0	8	11
A	0	0	1	0	4	7	10	5	4	**7**	1	5	4
G	0	5	0	0	5	9	3	6	10	**9**	3	0	1
C	0	0	10	3	0	2	5	0	3	6	**14**	7	0
A	0	0	3	6	0	0	7	10	3	0	7	**19**	12
C	0	0	5	0	2	0	0	3	6	0	5	12	15
T	0	0	0	10	3	0	0	0	0	2	0	5	17

TAB. 3.12 – *Exemple d'alignement local*

En remontant jusqu'à la dernière valeur strictement positive nous obtenons le résultat suivant :

G A A G – G C A
G C A G A G C A

3.6 La recherche de similarité dans les banques de séquences

Les méthodes d'alignements présentées précédemment sont efficace pour la comparaison exacte de deux séquences. Toutefois elles ne sont pas adéquates pour la recherche de similarité dans les banques de séquences biologique tel que GenBank [23]. Ainsi, pour pouvoir déterminer à partir d'une séquence toutes les séquences de la banque qui lui sont similaires, les algorithmes précédents sont trop gourmands en ressources. Pour contourner cet obstacle des heuristiques ont été proposées. Actuellement, les programmes de recherche de similarité dans les banques de séquences les plus utilisés sont probablement FASTA et BLAST [34].

3.6.1 FASTA

FASTA [35] (FAST All) est basé sur l'identification rapide des mots communs entre la séquence à analyser et les séquences de la banque. La taille du mot recherché est un paramètre que l'on peut choisir entre 1 et 6 (2 pour les protéines, 4 à 6 pour l'ADN). FASTA cherche les mots en se basant sur la technique de *Dotplot*. La *Dotplot*, aussi appelée *Dot matrix*, est une méthode graphique qui consiste à mettre une séquence horizontalement et l'autre verticalement puis pour chaque identité entre les bases on met un point (figure 3.2).

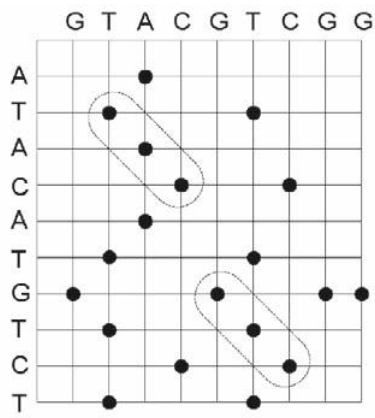

FIG. 3.2 – *Représentation en Dotplot de deux séquences*

A l'aide de cette technique FASTA identifie les diagonales ayant le plus grand score et

en prenant en considération les pénalités du mismatch (Figure 3.3 étape 1) . Les scores des dix meilleures diagonales vont être recalculés en utilisant une matrice de score comme *PAM* ou *BLOSUM* (Figure 3.3 étape 2). Les alignements dans la même diagonale vont êtres sélectionnées et rattachées pour former un seul alignement contenant des brèches (Figure 3.3 étape 3) .

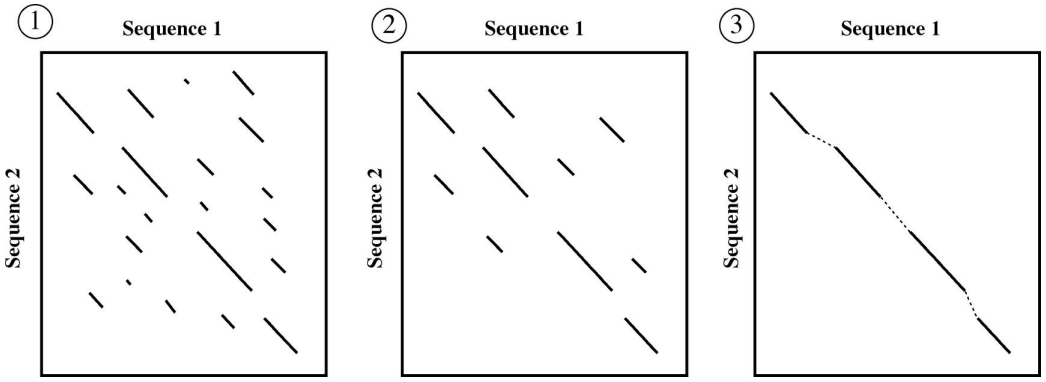

FIG. 3.3 – *Identification de la similarité des séquences avec FASTA*

Ce sont les scores des diagonales qui vont être utilisés pour classer les séquences de la base. Les recherches peuvent être optimisé en appliquant un des algorithmes *Needleman-Wunsch* ou *Smith-Waterman* mais seulement sur la zone contenant les dix diagonales.

3.6.2 BLAST

BLAST [36] (Basic Local Alignment Search Tool) est actuellement le programme le plus utilisé pour la recherche de similarité dans les banques d'ADN et de protéines [23].

Soit S une séquence donnée pour laquelle il faut trouver toutes les séquences de la base qui lui sont similaires. L'algorithme peut être décomposé en trois étapes principales :

- Dans un premier temps, l'algorithme va chercher toutes les sous-séquences de S de taille w. La taille par défaut est de 11 pour l'ADN et de 3 pour les protéines. Chacune des sous-séquences est comparée aux sous-séquences de taille w des séquences de la banque de données. L'algorithme crée alors une liste des sous-séquences de S dont la comparaison donne un score supérieur à une valeur fixée.
- La deuxième étape reprend la liste précédente, ainsi que la ou les séquences de la banque de données qui lui sont liées. Pour chaque paire ainsi formée, l'algorithme cherche à étendre au maximum la longueur des sous-séquences. La limite de cette extension est fixée par un seuil au-dessous duquel il ne faut pas aller.

– Les alignements locaux qui ont ainsi été obtenus ne sont pas tous de la même qualité. Dans cette dernière étape, BLAST ordonne les résultats obtenus en fonction de leur score.

3.7 Conclusion

L'alignement par paires est un composant fondamental pour de nombreuses applications de la bioinformatique. Il est extrêmement utile dans l'analyse structurelle, fonctionnelle et évolutionnaire des séquences. En plus, comme nous le verrons au prochain chapitre, il constitue bien souvent une étape nécessaire pour de nombreux algorithmes d'alignement multiple.

Contrairement à la recherche exacte de motifs présentée au chapitre précédent, l'alignement de séquences utilise la notion de distance et de similarité entre les séquences. Ces Notion ont été détaillées en premier lieu dans ce chapitre en présentant quelques types de distances, ainsi que les fonctions d'évaluation pour les alignements par paires. Ces dernières utilisent généralement une matrice de substitution et un modèle d'évaluation pour les brèches.

Ensuite, nous avons donné les différentes stratégies d'alignement de séquences en commençant par l'alignement global. Il s'agit de trouver le meilleur alignement possible des deux séquences complètes. Nous avons présenté par la suite l'algorithme de Needleman-Wunsch qui est un algorithme efficace pour la résolution du problème d'alignement global basé sur la programmation dynamique.

Aussi, nous avons présenté l'alignement local qui consiste à trouver l'alignement dont l'évaluation est la meilleure parmi les alignements de toutes les sous-séquences et pas les deux séquences complètes. Comme pour l'alignement global, il existe un algorithme basé sur la programmation dynamique efficace pour la résolution du problème d'alignement local qui est l'algorithme de Smith-Waterman.

Nous avons terminé ce chapitre en présentant les deux programmes de recherche de similarité dans les banques de séquences les plus utilisés qui sont : FASTA et BLAST.

Chapitre 4

Alignement multiple de séquences

4.1 Introduction

Le but de la comparaison de séquences protéiques est de découvrir des similitudes biologiques (i.e. structurelles ou fonctionnelles) parmi les protéines. Des protéines biologiquement similaires peuvent ne pas exhiber une forte similitude de séquences et l'on aimerait reconnaître la ressemblance structurelle/fonctionnelle, même lorsque les séquences sont très différentes. Si la similitude de séquences est faible, l'alignement par paires peut ne pas identifier des séquences apparentées biologiquement, car de faibles similitudes au niveau des paires peuvent faire échouer les tests statistiques. La comparaison simultanée de nombreuses séquences permet souvent de trouver des similitudes invisibles dans la comparaison de séquences par paires [37].

Dans ce chapitre nous allons commencé par définir l'alignement multiple de séquences et donner quelques fonctions d'évaluations utilisées. Par la suite, nous allons donné une classification des algorithmes d'alignement multiple de séquences en présentant le principe de quelques algorithmes de chaque catégorie. Après cela, nous allons donné un récapitulatifs des algorithmes présentés et nous terminons le chapitre par la présentation des benchmarks d'alignement multiple de séquences.

4.2 Définitions

L'alignement multiple de séquences MSA (Multiple Sequence Alignment) consiste à aligner simultanément plusieurs séquences (voir la figure 4.1). Le problème vu au chapitre

précédent peut être généralisé pour un ensemble de k séquences, avec $k > 2$. Les définitions ainsi que les propriétés vont également pouvoir être généralisées.

Définition 4.1 (Alignement Multiple de Séquences) Soit $S = S_1, S_2, ..., S_k$ un ensemble de k séquences définies sur un alphabet Σ, et soit $\Sigma' = \Sigma \cup \{-\}$, où "$-$" est le symbole pour représenter une brèche dans une séquence. Un alignement S est un ensemble $S' = \{S'_1, S'_2, ..., S'_k\}$ de séquences sur Σ', satisfaisant les trois propriétés suivantes :

- Les séquences de S' sont toutes de la même longueur n,
- Pour tout entier i de $[1..k]$, $ote(S'_i) = S_i$,
- Aucune colonne n'est constituée uniquement de brèches.

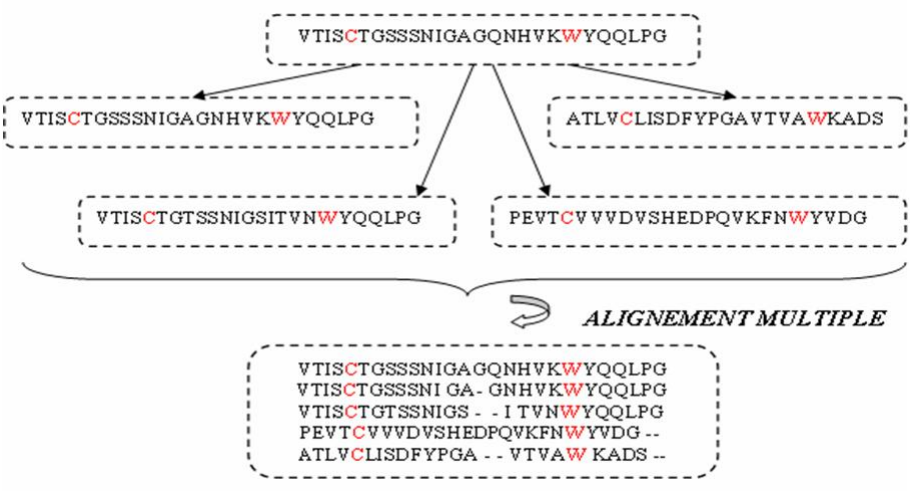

FIG. 4.1 – *Exemple d'un alignement multiple de séquences protéiques*

Le problème d'alignement multiple de séquences peut être défini de façon équivalente au moyen d'une matrice k x n. Chaque séquence de S' est placée sur une ligne de cette matrice. La dernière condition de la définition peut alors devenir : il n'existe pas de colonne de la matrice constituée uniquement de "$-$" [1].

En restreignant cette définition à $k = 2$, nous obtenons bien une définition équivalente à celle de l'alignement par paire.

4.3 Les utilisations en bioinformatique

L'alignement multiple de séquences permet de mettre en évidence les similarités entre plusieurs séquences. Il est donc possible de comparer simultanément la proximité de toutes ces séquences. Les informations apportées par ces comparaisons permettent d'obtenir des

renseignements importants sur les séquences comme les distances d'une séquence par rapport aux autres ou encore la mise en évidence de zones identiques entre plusieurs ou toutes les séquences.

Or ces opérations sont très employées en bioinformatique pour la résolution de plusieurs problèmes. L'alignement multiple de séquence est donc principalement utilisé comme opération préalable pour ces différents problèmes. Citons par exemple la construction de phylogénie, le prédiction de structure 3D et la détermination de fonction des protéines [1].

4.4 Les fonctions d'évaluation

Il existe plusieurs fonctions d'évaluation des alignements multiples, nous présentons dans ce qui suit celles qui sont les plus connues.

4.4.1 La somme des paires (Sum of Pairs : *SP*)

La fonction permettant d'évaluer un alignement de deux séquences données au chapitre précédent peut être généralisée afin d'évaluer un alignement multiple. Cette fonction est appelée la somme des paires (SP), c'est l'une des méthodes les plus utilisées.

Définition 4.2 (Somme des paires) Soit S un ensemble de k séquences, et soit A un alignement multiple de S de longueur l. Soit f une fonction permettant d'évaluer un couple de résidus, on définit la fonction de somme des paires par :

$$SP(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{p=1}^{l} f(S_i(p), S_j(p))$$

Cette définition de Sp dépend de la fonction f permettant d'évaluer une paire de résidus, et donc elle dépend de la matrice de substitution utilisée ainsi que de la méthode utilisée pour évaluer les brèches.

En associant la valeur 0 au couple $(-,-)$, la définition donnée ci-dessus peut être réécrite sous forme d'une somme d'évaluations d'alignements de deux séquences :

$$SP(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} Eval(A(i,j))$$

Où $A(i,j)$ correspond à l'alignement des deux séquences S_i et S_j .

Exemple : Soit l'alignement multiple suivant :

$$\begin{array}{ll} S_1 : & \text{A G C T A A} - \text{A} \\ S_2 : & \text{A} - \text{C T A A T A} \\ S_3 : & \text{A} - - \text{T C A T A} \end{array}$$

Et soit **M** la matrice de substitution utilisée :

M	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

Les brèche sont à coût constant -1 . La fonction SP commence par évaluer S_1 avec S_2, S_1 avec S_3 puis S_2 avec S_3 comme le montre les figures suivantes :

S_1 :	A	G	C	T	A	A	–	A
S_2 :	A	–	C	T	A	A	T	A
28	5	-1	5	5	5	5	-1	5

S_1 :	A	G	C	T	A	A	–	A
S_3 :	A	–	–	T	C	A	T	A
13	5	-1	-1	5	-4	5	-1	5

S_2 :	A	C	T	A	A	T	A
S_3 :	A	–	T	C	A	T	A
20	5	-1	5	-4	5	5	5

$SP = 28 + 13 + 20 = 61$ et donc l'alignement précédent a un score de 61.

Remarque : la valeur 0 a été attribuée au couple $(-,-)$ qui n'a pas été représenté dans le cas de (S_2,S_3).

4.4.2 La somme des paires pondérées (Weighted Sum of Pairs : WSP)

La fonction SP présentée précédemment est simple à mettre en œuvre et elle est tout à fait adaptée lorsque toutes les séquences sont très similaires. En revanche, lorsque ce n'est pas le cas cela peut poser des problèmes [38]. L'évaluation par la fonction de somme des paires est biaisée dès qu'il y a un ou plusieurs sous-ensembles de séquences très similaires. Il existe donc une variante permettant de corriger en partie ce défaut. Pour cela des poids

sont attribués aux séquences pour diminuer ce biais en donnant plus de poids aux séquences distantes pour qu'elles deviennent plus importantes.

Définition 4.3 (Somme des paires pondérée) Soit S un ensemble de k séquences, soit A un alignement multiple de S de longueur l, et soit w_i le poids la séquence S_i. On définit la fonction de somme des paires pondérée par :

$$WSP(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_i.w_j.Eval(A(i,j))$$

La détermination du poids w_i pour chaque séquence dépend de sa distance par rapport aux autres. Les valeurs doivent donc être déterminées pour chaque problème. La méthode généralement utilisée est basée sur l'algorithme du Neighbour-Joining [39].

4.4.3 La fonction *Coffee*

L'algorithme *Coffee* (Consistency-based Objective Function For alignmEnt Evaluation) [40] propose une méthode différente pour évaluer les alignements multiples.

Soit $S = \{S_1, S_2, ..., S_k\}$, un ensemble de $k > 2$ séquences, et soit $A = \{S'_1, S'_2, ..., S'_k\}$ un alignement multiple de S. L'évaluation au moyen de la fonction *Coffee* peut se décomposer en deux étapes :

- Dans un premier temps, l'algorithme commence par réaliser tous les alignements par paires des séquences de S,
- L'évaluation se fait au moyen des alignements obtenus à la première étape. En effet, l'alignement par paire A_{ij} de chaque couple de séquences (S'_i, S'_j) est comparé à l'alignement par paire S_{ij} des séquences S_i et S_j.

Cette évaluation se fait en comparant les paires de résidus de A_{ij} et S_{ij} . Ainsi, l'alignement A_{ij} est parcouru, et pour chaque paire de résidus, une valeur est attribuée. Si cette paire de résidus est présente dans l'alignement S_{ij} elle a la valeur 1, sinon elle a la valeur 0. L'évaluation de A_{ij} se fait en additionnant les valeurs de chaque paire de résidus, celle-ci est notée $SCORE(A_{ij})$.

L'évaluation de A au moyen de la fonction *Coffee* est obtenue en additionnant les valeurs de tous les couples de séquences A_{ij} de A. Pour pallier le problème de la fonction de somme des paires, les valeurs sont ici aussi pondérées. Nous obtenons donc la valeur suivante :

$$f(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_i.w_j.SCORE(A(i,j))$$

Cette fonction permet de comparer les valeurs associées à plusieurs alignements possibles de S, mais elle ne permet pas de connaître la qualité réelle de ces alignements. Ainsi la longueur des séquences a autant d'influence sur la valeur de cette fonction que la qualité de l'alignement. Pour éviter cela, *Coffee* est définie à partir de la fonction donnée ci-dessus, en la divisant par la somme des longueurs des alignements par paires de A :

$$Coffee(A) = \frac{\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_i.w_j.SCORE(A(i,j))}{\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_i.w_j.Long(A(i,j))}$$

Où *Long* est la fonction qui associe a chaque alignement de séquences sa longueur

Si toutes les paires de résidus de l'alignement multiple A sont identiques aux paires de résidus des alignements par paires correspondants, cela veut dire que pour tout i et j $SCORE(A_{ij}) = Long(A_{ij})$. Donc on obtient $Coffee(A) = 1$.

A l'opposé, si aucune paire de résidus de A_{ij} n'est présente dans l'alignement S_{ij} quelques soient i et j, on obtient $Coffee(A) = 0$. *Coffee* fournit donc comme évaluation un pourcentage de similitude entre les alignements par paires des séquences de S et les paires de séquences alignées de A. Comme l'hypothèse de départ était que les alignements par paires des séquences de S sont tous exacts, plus la valeur de $Coffee(A)$ est proche de 1, meilleur est l'alignement A.

4.5 Classification des algorithmes

Le problème d'alignement multiple de séquences a été démontré NP-Complet [41]. Pour cela, plusieurs méthodes utilisant différentes stratégies ont été proposées pour résoudre ce problème. Il est toutefois possible de regrouper les algorithmes d'alignement multiples de séquences selon trois classes [10] en fonction de critères assez simples.

Nous avons les algorithmes exacts (basés sur la programmation dynamique) permettant de réaliser des alignements d'un petit nombre de séquences. Les algorithmes approchés sont bien sûr beaucoup plus nombreux, et ils peuvent être également subdivisés en deux catégories : les algorithmes progressifs et les algorithmes itératifs.

Les algorithmes progressifs ont tous un point commun lié au processus d'alignement. Les séquences sont alignées progressivement en suivant un ordre défini, seule la façon dont vont être alignés les groupes de séquences va différer. Pour les algorithmes itératifs toutes les séquences (ou presque) sont alignées simultanément et les méthodes sont en revanche très

différentes les unes des autres.

Remarque : il existe dans la littérature d'autres classifications, citons comme exemple les algorithmes stochastiques et les algorithmes déterministes, les méthodes d'alignement multiples globales et les méthodes locales.

4.6 Les algorithmes exacts

Nous avons présenté au chapitre précédent une méthode basé sur la programmation dynamique capable de résoudre le problème d'alignement de deux séquences. Cette méthode peut être généralisé pour un nombre supérieur à deux séquences.

4.6.1 Algorithme basé sur la programmation dynamique

Cas de 3 séquences

Le problème consiste ici à aligner 3 séquences S, T et U de longueurs respectives p, q et r. En faisant le parallèle avec le cas de 2 séquences, nous appellerons $P(i, j, k)$ le problème consistant à aligner les 3 sous-séquences $S[1..i]$, $T[1..j]$ et $U[1..k]$, et la valeur associée à ce problème sera notée $V(i, j, k)$. Aligner S, T et U consiste donc à déterminer $P(p, q, r)$ et la valeur de cet alignement est $V(p, q, r)$. Dans ce qui suit nous allons détaillé l'algorithme de l'alignement avec brèches à coût constant.

Le principe de l'algorithme est le même que pour deux séquences, il suffit de chercher à déterminer la dernière position de l'alignement. Plusieurs cas sont alors possibles, selon qu'il y a une, deux ou trois lettres.

- La dernière position de l'alignement constituée d'une unique lettre peut se produire de trois façons différentes, selon la séquence où se trouve la lettre.
- La dernière position de l'alignement constituée de deux lettres peut se produire de trois façons différentes, selon la séquence où se trouve la brèche.
- La dernière position de l'alignement constituée de trois lettres correspond à un cas unique.

Il y a donc au total 7 possibilités pour la dernière position de l'alignement. Ces 7 possibilités correspondent aux 7 sous-problèmes directs de $P(p, q, r)$. La valeur de $V(p, q, r)$ en fonction de ces sous-problèmes est donnée par l'équation suivante :

$$V(p,q,r) = max \begin{pmatrix} V(p-1,q,r) + 2.val('s','-'), \\ V(p,q-1,r) + 2.val('t','-'), \\ V(p,q,r-1) + 2.val('u','-'), \\ V(p-1,q-1,r) + val('s','t') + val('s','-') + val('t','-'), \\ V(p-1,q,r-1) + val('s','u') + val('s','-') + val('u','-'), \\ V(p,q-1,r-1) + val('t','u') + val('t','-') + val('u','-'), \\ V(p-1,q-1,r-1) + val('s','t') + val('s','u') + val('t','u') \end{pmatrix}$$

Comme pour le cas de l'alignement de deux séquences, le principe de la programmation dynamique peut être utilisé pour résoudre le problème. Toutefois dans ce cas il n'est plus possible d'utiliser une matrice en deux dimensions puisqu'il faut conserver toutes les valeurs $V(i,j,k)$. Il est donc nécessaire d'utiliser un cube de taille $(p+1)(q+1)(r+1)$.

L'algorithme se termine lorsque $V(p,q,r)$ a été calculée. Une fois cette valeur obtenue, il est possible de construire l'alignement correspondant en utilisant la même méthode que pour deux séquences. Il suffit pour cela de parcourir la matrice jusqu'à l'origine afin de déterminer quelle suite d'opérations d'édition a été utilisée. Le résultat que l'on obtient est similaire à ce que l'on peut obtenir pour deux séquences, mais en 3 dimensions. La figure 4.2 montre un exemple de parcours de la matrice pour construire l'alignement.

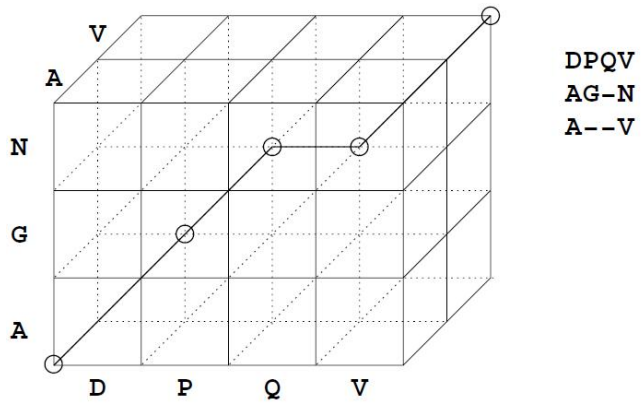

FIG. 4.2 – *Exemple de construction de l'alignement pour 3 séquences*

Cas général

Le principe d'alignement exposé pour deux et trois séquences peut être généralisé à un nombre de séquences $n > 3$ quelconque. Le principe reste le même, il est nécessaire d'utiliser

une matrice en dimension n, dont la taille est le produit cartésien des longueurs de chacune des séquences.

L'algorithme pour remplir cette matrice est là encore basé sur le principe de la programmation dynamique. Les cas de base sont constitués par les n alignements possibles de $n-1$ séquences. Cette matrice peut être progressivement remplie, jusqu'à l'obtention de la valeur du meilleur alignement. La construction de l'alignement correspondant s'obtient en remontant jusqu'au point origine de la matrice [1].

Les limites de la programmation dynamique

Le principe de l'algorithme basé sur la programmation dynamique est assez simple, mais en revanche il est très difficilement utilisable sur les ordinateurs actuels. Dans [1] une étude a été faite qui permet de donner une approximation de la mémoire nécessaire pour la construction de la matrice lors de l'utilisation du programme (tableau 4.1). Il s'agit de la quantité de mémoire minimum, c'est-à-dire dans le cas où l'implémentation est faite de façon à optimiser ce critère. Pour ce faire, il ne faut pas conserver toutes les valeurs de la matrice, mais uniquement les directions pour la construction de l'alignement.

	$l = 100$	$l = 200$	$l = 300$	$l = 400$	$l = 500$
$n = 4$	100 Mo	1.6 Go	8.1 Go	25.6 Go	62.5 Go
$n = 5$	10 Go	320 Go	2.4 To	10.2 To	31.2 To
$n = 6$	1 To	64 To	729 To	4.1 Po	15.6 Po
$n = 7$	100 To	12.8 To	218.7 Po	1.6 Eo	7.8 Eo
$n = 8$	10 Po	2.5 Eo	65.6 Eo	655 Eo	3.9 Zo

TAB. 4.1 – *Mémoire nécessaire pour un alignement multiple*[1].

D'après le tableau 4.1 nous constatons que la méthode d'alignement basée sur la programmation dynamique ne peut être utilisée, dans les ordinateurs actuels, que pour aligner un petit nombre de séquences de petite taille. Comme nous allons le voir, cette méthode est tout de même utilisable pour aligner plus de séquences si elle est couplée à des heuristiques.

4.6.2 Algorithme *MSA*

Cas de 2 séquences

L'algorithme *MSA* [42] propose une heuristique basée sur l'algorithme complet de *Needleman-Wunsch*. Le principe est simple, et il est facile à comprendre sur un alignement de deux séquences. En pratique on peut remarquer que le chemin permettant la construction de l'alignement des deux séquences se situe à proximité de la diagonale (figure 4.3).

En fonction de la similarité entre les séquences à aligner, il est possible de déterminer avant d'utiliser l'algorithme de programmation dans quelle partie se trouvera l'alignement. Plus la similarité est forte, plus la zone centrale peut être réduite.

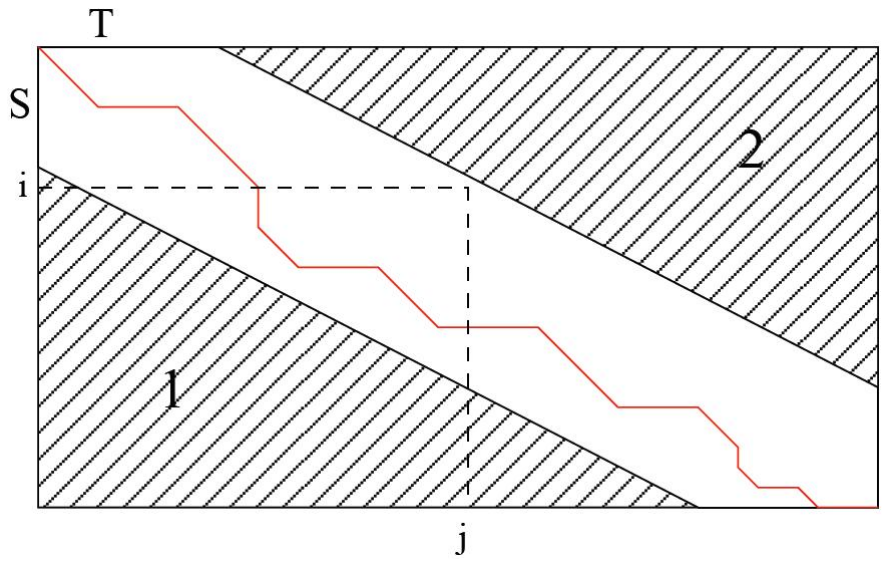

FIG. 4.3 – *Restriction pour un alignement de deux séquences.*

Sur la figure 4.3, les zones hachurées marquées 1 et 2 correspondent aux valeurs qui n'ont pas besoin d'être calculées. Même si ces valeurs ne sont pas connues, l'algorithme basé sur la programmation dynamique peut être facilement modifié pour le calcul des valeurs situées à la limite de ces zones. Deux cas sont possibles suivant la zone considérée :

– La limite supérieure de calculs est marquée par la zone 2, et la formule donnée précédemment n'est plus utilisable. Par exemple pour la coordonnée (i, j), $V(i-1, j)$ n'est pas définie. Or par hypothèse, nous savons que l'alignement ne peut pas être dans la zone 2, il est donc impossible que $V(i, j)$ soit obtenue à partir de cette valeur. Pour la zone limitrophe supérieure, la formule devient donc :

$$V(i,j) = max \begin{pmatrix} V(i,j-1) + val('-','t'), \\ V(i-1,j-1) + val('s','t') \end{pmatrix}$$

– De même pour la zone limitrophe inférieure, la formule devient :

$$V(i,j) = max \begin{pmatrix} V(i-1,j) + val('s','-'), \\ V(i-1,j-1) + val('s','t') \end{pmatrix}$$

Les formules données ici correspondent à un alignement avec coût constant pour les brèches. La formule utilisée pour les alignements avec coût affine peut bien sûr être simplifiée de la même façon.

Cas de $n > 2$ séquences

La méthode exposée pour 2 séquences est généralisable pour un nombre quelconque n de séquences. La zone de calculs est toujours une partie de la matrice centrée sur la diagonale issue de l'origine.

Cette méthode basée sur une méthode exacte est une heuristique. Même s'il est fortement probable qu'elle donne une solution optimale, cela n'est aucunement garanti. Elle offre l'avantage de permettre d'augmenter la limite du nombre de séquences pouvant être alignées.

4.6.3 Algorithme *DCA*

L'algorithme *MSA* est basé sur une restriction de la zone de calculs autour de la diagonale de la matrice. Il est à la base d'un autre algorithme de type "divide-and-conquer" appelé *DCA* (Divide and Conquer mulitple sequence Alignment)[43].

Le principe consiste à découper les séquences en sous ensembles de segments. Ces derniers doivent être aussi petits pour qu'ils puissent être traités par la méthode *MSA*. Les sous alignements produits sont ensuite rassemblés par l'algorithme DCA (figure 4.4).

Nous constatons donc qu'avec deux heuristiques, il est possible d'utiliser la programmation dynamique pour obtenir des alignements comportant nettement plus de séquences. Rien ne garantit toutefois que l'on peut calculer l'optimum.

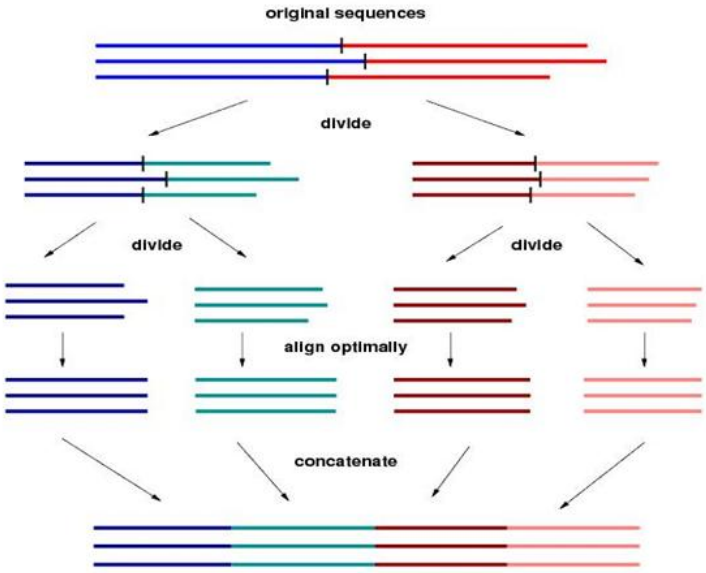

FIG. 4.4 – *Les étapes de l'algorithme DCA.*

4.7 Les algorithmes progressifs

Comme l'indique leur nom, les algorithmes progressifs consistent à réaliser les alignements multiples en alignant progressivement les séquences. D'une façon générale, tous les algorithmes progressifs reposent sur le même principe : commencer par aligner des sous-groupes de séquences puis essayer de les combiner entre eux pour former des alignements contenant de plus en plus de séquences. L'algorithme s'arrête lorsque toutes les séquences ont été regroupées pour former l'alignement multiple complet.

La première description d'un algorithme progressif a été faite par *Hogeweg et Hesper* [44], le but était alors de produire un alignement multiple destiné à réaliser une phylogénie sur l'ensemble des séquences. Il s'agit bien d'un algorithme respectant les principes donnés précédemment, mais simplifié à l'extrême [1].

La première amélioration importante est décrite dans l'algorithme de *Feng et Doolittle* [45]. Il utilise la notion de *profil* pour réaliser les alignements ainsi que le calcul d'un arbre appelé *guide-tree* pour indiquer l'ordre dans lequel il convient d'aligner les séquences. La majorité des algorithmes progressifs actuels sont en très grande partie basé sur l'algorithme de *Feng et Doolittle*.

Description d'un profil

Un profil est une séquence virtuelle formée à partir d'un alignement. Il s'agit d'une séquence consensus qui est la plus proche possible de l'ensemble des séquences alignées [1].

Ainsi, pour réaliser l'alignement de deux groupes de séquences alignées, la méthode propose d'aligner les profils. Une fois que les profils ont été alignés au moyen de l'algorithme de *Needleman-Wunsch*, il ne reste plus qu'à les remplacer par les alignements auxquels ils correspondent. Pour cela il faut uniquement faire intervenir les brèches ayant été insérées dans chaque profil.

Soient A_1 et A_2 deux alignements de séquences et soient p_1 et p_2 leurs profils respectifs. En alignant p_1 et p_2, des brèches se trouvent insérées, et il faut donc au final les insérer également dans A_1 et A_2. A toute brèche insérée dans un profil correspond une colonne complète de brèches dans son alignement, donnant ainsi deux alignements A'_1 et A'_2 de même longueur. En regroupant A'_1 et A'_2, on obtient l'alignement de A_1 et A_2.

Aligner une séquence avec un alignement correspond à un cas particulier du problème précédent. Il suffit dans ce cas d'aligner la séquence avec le profil de l'alignement. La construction du résultat final s'obtient de la même façon que précédemment.

Construction d'un profil

La construction d'un profil nécessite de remplacer une colonne de l'alignement par une lettre unique. Pour cela des règles assez simples ont été établies. Deux cas doivent être envisagés :

- si la colonne est composée uniquement par une seule lettre ou des brèches, alors cette lettre sera conservée dans le profil,
- si la colonne est composée de plusieurs lettres différentes, il faut prendre en compte la probabilité d'apparition de chacune des lettres. Cette probabilité est multipliée par le nombre d'occurrences de cette lettre dans la colonne. La lettre avec la valeur la plus forte est conservée dans le profil de l'alignement.

Création d'un *Guide-Tree*

Afin d'indiquer l'ordre dans lequel il convient d'aligner les séquences nous utilisons un *guide-tree* (arbre guide) où les feuilles représentent les séquences initiales, les nœuds sont les différents profils obtenus, et la racine correspond au résultat de l'alignement multiple (figure 4.5).

Pour avoir un alignement multiple de qualité nous avons besoin de construire un arbre dont l'ordre d'alignement auquel il correspond est le plus pertinent possible. Différentes stratégies peuvent être envisagées pour la création de cet arbre. Le seul critère disponible pour comparer les séquences est la similarité. Or un des critères de qualité pour un alignement est

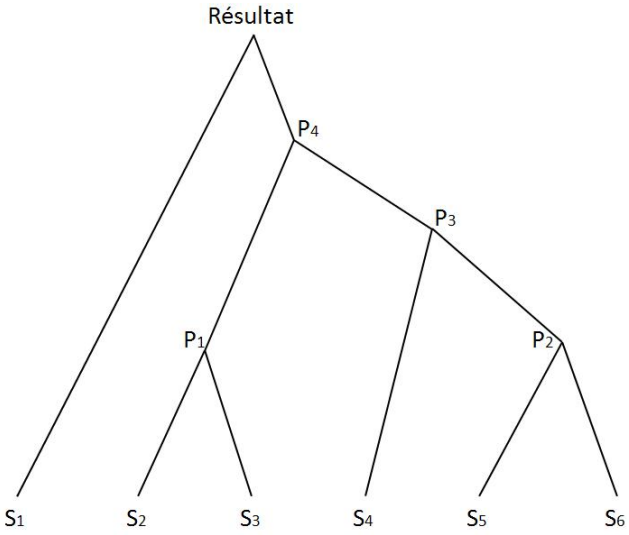

FIG. 4.5 – *Exemple de guide-tree pour 6 séquences avec création de 4 profils p_1, p_2, p_3 et p_4.*

de réussir à maximiser le nombre de correspondances en minimisant le nombre de brèches. Il est donc préférable de commencer par aligner ensemble les séquences similaires afin d'insérer le moins possible de brèches.

La construction d'un *guide-tree* nécessite donc de définir des distances entre toutes les séquences à aligner. Pour cela, tous les alignements par paires doivent d'abord être calculés. L'évaluation traditionnelle de ces alignements au moyen d'une matrice de substitution n'est pas très pertinente car il ne s'agit pas ici de déterminer le meilleur alignement de deux séquences. Or cette méthode ne permet aucunement de comparer des alignements de séquences différentes. Les séquences n'étant pas les mêmes, et les longueurs pouvant être très différentes, il est nécessaire de prendre cela en compte.

Nous avons la distance d entre les deux séquences est définie comme étant $d = 1 - s$. tel que s est la similarité définie au chapitre précédent. Cette valeur s vaut 1 lorsque les deux séquences sont identiques, et sera d'autant plus proche de 0 si les séquences sont différentes.

En calculant les distances entre toutes les séquences, il est donc possible d'utiliser un des algorithmes permettant la création d'un arbre. Nous citons ici les deux algorithmes les plus connus :

- UPGMA (Unweighted Pair Group Method with Arithmetic mean) [46]
- Neighbour-Joining [39]

Voici un schéma qui illustre le principe général des algorithmes progressifs (figure 4.6).

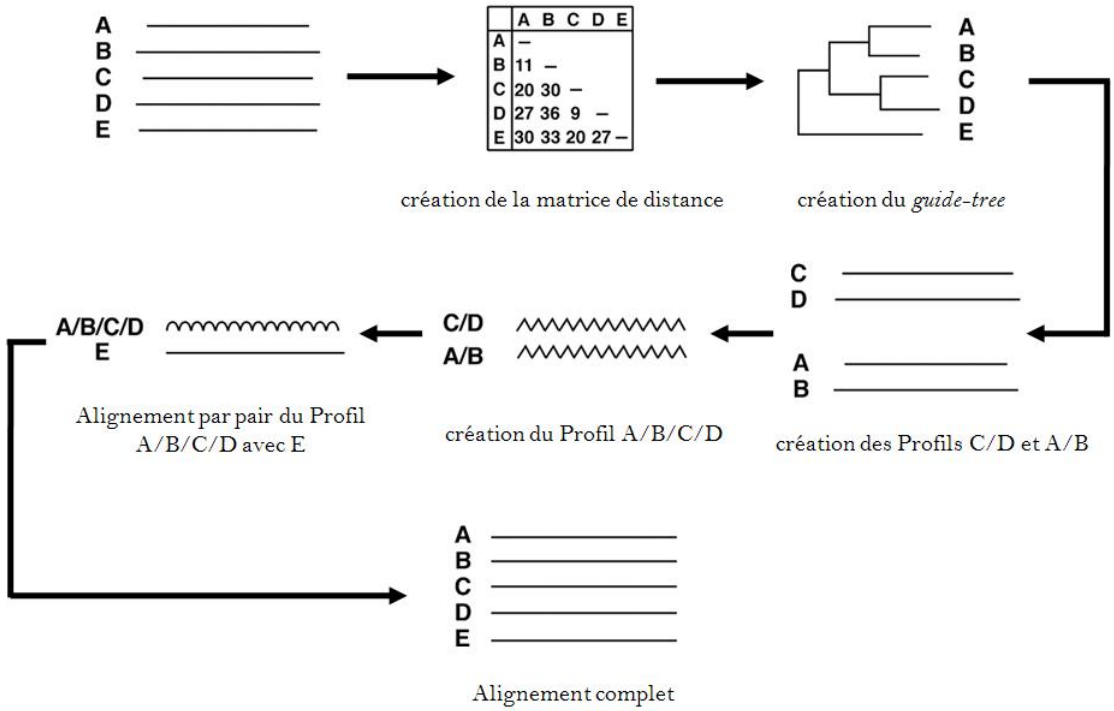

FIG. 4.6 – *Les différentes étapes des algorithmes progressifs .*

Même si les algorithmes progressifs donnent en général de bons résultats dans un temps minimal, cela n'empêche qu'ils ont quelques inconvénients comme :

- La solution est très dépendante du premier alignement de paires. C'est-à-dire que le mauvais choix des premières séquences à aligner va donner des alignements de mauvaises qualités.
- La propagation des erreurs produites dans les premiers alignements aux alignements suivants.
- Le choix des paramètres a une grande influence sur la qualité du résultat. Il n'existe pas une méthode universelle pour régler les paramètres d'alignements tels que les pénalités des gaps et la matrice de substitutions utilisée.

Il existe un nombre important d'algorithmes progressifs, nous présentons dans ce manuscrit quelques algorithmes fréquemment utilisés.

4.7.1 *Clustal W*

Clustal W [47] est basé sur le principe de l'algorithme de *Feng et Doolittle*. Le principe général est le suivant :

1. Calculer tous les alignements par paires des séquences, et en déduire une matrice des

distances.

2. Construire un guide-tree à partir de la matrice des distances en utilisant la méthode du *Neighbour-Joining*.
3. Utiliser le guide-tree afin de déterminer l'ordre dans lequel les séquences doivent être alignées :
   (a) Choisir les séquences ou profils à aligner en suivant le *guide-tree*.
   (b) Les aligner en utilisant une méthode basée sur la programmation dynamique.
   (c) Créer un profil à partir du résultat de l'alignement.
   (d) Si on n'est pas arrivé à la racine de l'arbre reprendre en 3.a.
4. Retourner l'alignement obtenu.

Clustal W fait intervenir des paramètres supplémentaires dans la phase d'alignement de deux séquences et/ou profil citons :

- *Clustal W* prend en compte les propriétés physico-chimiques propres aux différents acides aminés. Ainsi si deux acides aminés sont normalement plus difficiles à séparer, un surcoût sera attribué pour l'insertion d'une brèche.
- *Clustal W* détermine les paramètres d'alignement à utiliser (coût d'ouverture et d'extension de brèches ainsi que matrice de substitution) en fonction des séquences à aligner.

A la fin des années 1990, *Clustal W* étais le programme d'alignement multiple le plus utilisé [48]. Il donne de bons résultats dans la plupart des cas, ce qui le rend encore très utilisé aujourd'hui.

Remarque : Dans *Clustal W*, le *guide-tree* a été calculé en utilisant la méthode *Neighbour-Joining*. Dans les nouvelles versions du programme, *UPGMA* est utilisée. Ce qui rend la nouvelle version plus rapide surtout pour un grand nombre de séquences [48].

4.7.2 *T-Coffee*

T-Coffee (Tree-based Consistency Objective Function for alignment Evaluation) [49] est un algorithme qui donne en général de bons résultats mais avec un temps d'exécution relativement long par rapport aux nouvelles méthodes d'alignement multiple.

La première étape d'alignements par paires de *Coffee* est maintenant une étape de prétraitement où tous les alignements pouvant être obtenus par la méthode de programmation dynamique sont calculés. Ainsi, l'algorithme calcule les alignements par paires globaux et

locaux pour toutes les paires de séquences produites par deux méthodes connues (*ClustalW* et *Lalign* de *FASTA*).

Les différentes étapes de l'algorithme *T-Coffee* sont :

- Produire des bibliothèques primaires des alignements.
- Déduire des poids de la bibliothèque
- Combiner les bibliothèques ensemble dans la bibliothèque primaire
- Extension de la bibliothèque.
- Employez la bibliothèque étendue pour l'alignement progressif.

Voici un schéma qui illustre le déroulement de l'algorithme *T-Coffee* (figure 4.7) :

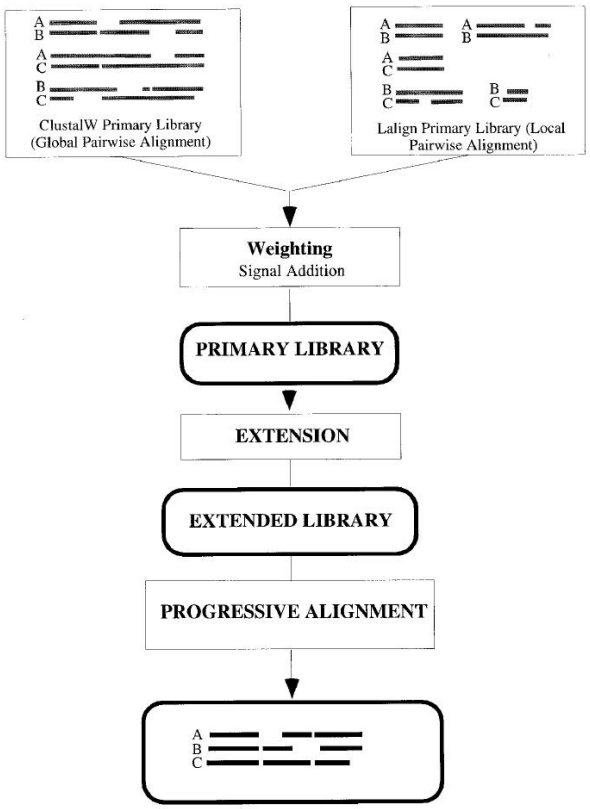

FIG. 4.7 – *Les différentes étapes de l'algorithme T-Coffee.*

Remarque : Il existe une extension de *T-Coffee* appelée *M-Coffee* [50], c'est une méta-méthode qui utilise les résultats de plusieurs autres algorithmes afin de déterminer la valeur d'un alignement. La version classique utilise 15 algorithmes représentatifs *(Clustal W, T-Coffee, ProbCons, ...)* pour créer 15 alignements de référence. Ces résultats sont utilisés pour l'évaluation de l'alignement multiple.

4.7.3 *MAFFT*

L'algorithme *MAFFT* [51] exploite les caractéristiques physico-chimiques des acides aminés qui composent les protéines pour établir le degré de similitude ou de divergence entre elles. Il utilise une *transformée de Fourier* rapide pour construire le *guide-tree* qui lui sert pour calculer l'alignement. Cette opération permet de réduire le temps nécessaire pour générer le *guide-tree*.

De plus, *MAFFT* apporte quelques modifications pour l'évaluation des alignements. La matrice de substitution utilisée est normalisée pour ne contenir que des valeurs positives. L'algorithme de *Needleman-Wunsch* donnant généralement de meilleurs résultats avec des matrices entièrement positives. Le coût des brèches est lui aussi adapté pour correspondre aux valeurs de la matrice de substitution qui a été calculée.

l'algorithme *MAFFT* donne de très bons résultats, et les temps de calculs sont également très faibles.

4.7.4 *MUSCLE*

L'algorithme *MUSCLE* [52] respecte le principe général des algorithmes d'alignement progressif, mais en apportant plusieurs modifications majeures. Pour obtenir le résultat, deux alignements sont réalisés, avec des méthodes différentes pour obtenir les deux *guide-trees*. Une phase de raffinement est ensuite proposée, pour vérifier la validité du second *guide-tree*.

MUSCLE utilise deux mesures de distance pour une paire de séquences : la distance *k-mer* [53] (pour les séquences non-alignées) et la distance *Kimura* [54] (pour les séquences alignées). *MUSCLE* utilise les *k-mer* pour créer la première matrice des distances. Il s'agit de toutes les sous-séquences de longueur k des alignements par paires qui s'apparient exactement. Cette mesure n'exige pas un alignement, elle donne un avantage significatif de vitesse contrairement à *Kimura* qui est plus précise mais nécessite un alignement .

L'algorithme de *MUSCLE* est divisé en trois grandes étapes (figure 4.8), chacune de ces étapes fournissant un alignement multiple des séquences.

Les différentes étapes de l'algorithme *MUSCLE* sont :

1. Construction du premier alignement :
   - Création d'une matrice D1 des distances *k-mer* entre toutes les paires de séquences.

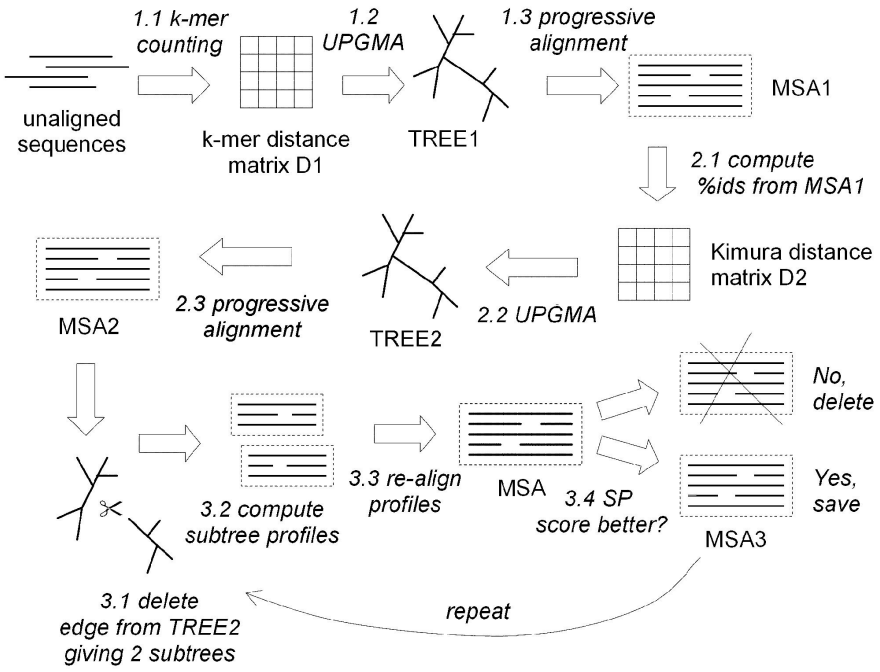

FIG. 4.8 – *Déroulement de l'algorithme MUSCLE.*

- Construction d'un *guide-tree* TREE1 avec la méthode de l'*UPGMA* à partir de la matrice D1.
- Alignements par paires des séquences ou des profils pour construire un alignement multiple MSA1.

2. Construction du second alignement :
   - Création d'une nouvelle matrice D2 de distances *Kimura* à partir de toutes les paires de séquences de MSA1.
   - Construction d'un second *guide-tree* TREE2 avec la méthode de l'*UPGMA* à partir de la matrice D2.
   - Alignements par paires des séquences ou des profils pour construire un alignement multiple MSA2.
3. Raffinement du résultat :
   - Choisir une branche de l'arbre TREE2.
   - Couper cette branche, et former les deux profils correspondants aux deux sous-arbres obtenus.
   - Aligner les deux profils pour former un nouvel alignement multiple de toutes les séquences.
   - Si l'alignement est meilleur il est conservé, sinon il sera rejeté.

– Ces opérations sont répétées jusqu'à stabilisation de la solution ou un nombre prédéfini de fois.

Muscle est actuellement un des meilleurs algorithmes d'alignements multiple, tant au point de vue de la qualité des résultats que de la rapidité.

4.7.5 *ProbCons*

ProbCons [55] est actuellement un des meilleur algorithme d'alignement multiple de séquences. il donne de très bons résultats en maintenant un temps de calcul acceptable.

ProbCons est basé sur *un modèle de Markov caché*. Ainsi, ce modèle permet de déterminer la probabilité $P(x_i \sim y_i|x, y)$ pour que le résidu x_i de la séquence x soit aligné avec le résidu y_i de la séquence y. Le *guide-tree* est construit à partir d'une matrice de distances calculée en utilisant ce modèle.

ProbCons est constitué de six étapes, les trois premières concernent le calcul des probabilités. Le reste des étapes est : construction d'un *guide-tree*, alignement et optimisation du résultat. Dans la phase d'optimisation, l'algorithme sépare l'alignement multiple en deux de façons aléatoires, et les réalignent. Ce processus est effectué un nombre fixe de fois. Lorsqu'un alignement de meilleure qualité est obtenu, il remplace l'alignement courant.

4.8 Les algorithmes itératifs

Contrairement aux algorithmes exacts et aux algorithmes progressifs, les algorithmes itératifs sont beaucoup plus variés dans les méthodes qu'ils utilisent pour réaliser l'alignement des séquences. Nous citons principalement *SAGA*, car cet algorithme reste une référence importante.

Les algorithmes itératifs sont souvent plus lents que les algorithmes progressifs car ils nécessites plus de calculs.

4.8.1 *SAGA*

SAGA (Sequence Alignment by Genetic Algorithm) [56] est basé sur un algorithme de type génétique dédié pour résoudre le problème d'alignement multiple de séquences. Une population G_0 d'alignements est initialement générée, et le programme permet de contrôler son

évolution. La population initiale est créée en générant cent individus par insertion aléatoire de brèches. Le pseudo-code de l'algorithme est le suivant :

Initialisation

1. Crée G_0

Évaluation

2. Évaluer la population de la génération n (G_n)
3. Si la population est stabilisée alors FIN
4. Choisir les individus à remplacer
5. Évaluer les descendants attendus

Reproduction

6. Choisir les parents à partir de G_n
7. Choisir l'opérateur
8. Générer le nouvel enfant
9. Garder ou abandonner le nouvel enfant dans G_{n+1}
10. Aller à 6 jusqu'a ce que tous les enfants sont mis dans G_{n+1} avec succès
11. $n = n + 1$
12. Aller à Évaluation

FIN

13. FIN

Il existe plusieurs opérateurs dans *SAGA* correspondant à des combinaisons ou à des mutations. Chaque opérateur est vu comme une fonction indépendante, prenant deux individus pour les combinaisons et un individu pour les mutations. la figure 4.9 donne un exemple d'une combinaison (croisement) entre deux alignements parents pour générer deux alignement enfants. le meilleur alignement enfant est choisi.

SAGA a la particularité de pouvoir optimiser n'importe quelle fonction objective. Plus tard, *SAGA* a été utilisé pour valider une fonction objective : *Coffee*. Les résultats sont considérés nettement meilleurs que ceux fournis par la première approche qui utilise *WSP*.

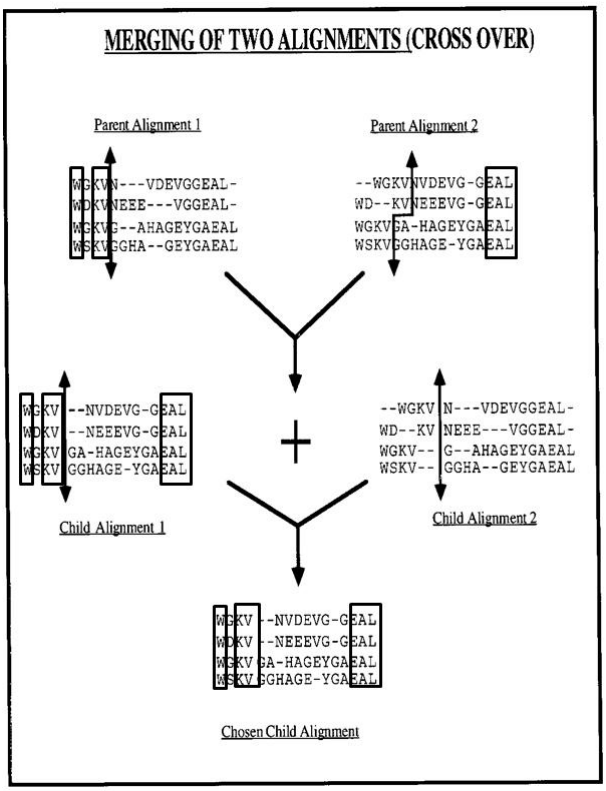

FIG. 4.9 – *Exemple d'un opérateur de combinaison (croisement) .*

4.8.2 Autres algorithmes

Nous citons ici quelques autres algorithmes qui montrent la diversité des méthodes utilisées par les algorithmes itératifs :

- Multiple alignment using hidden Markov models [57],
- A tabu search algorithm for post-processing multiple sequence alignment [58],
- Multiple Sequence Alignment Based on ABC_SA [59].

4.9 Récapitulatif des algorithmes

Nous proposons au tableau 4.2 un récapitulatif des algorithmes présentés précédemment. Pour chacun d'eux nous rappelons l'année de développement, de quel type d'algorithme il s'agit, ainsi que le principal avantage et problème de chaque algorithme.

Algorithme	Année	Classe	Avantages	Problèmes
MSA	1988	Exact	Le plus exact	Peut aligner un nombre très limité de séquences
DCA	1997	Exact	Peut aligner plus de séquences que MSA	Limité, requiert Plus de mémoire
Clustal W	1994	Progressif	Rapide, précision acceptable et simple à utiliser	Moins efficace que les nouveaux algorithmes
T-Coffee	2000	Progressif	Une bonne précision	Très lent par rapport aux nouveaux algorithmes
MAFFT	2002	Progressif	Très rapide, une bonne précision	Moins précis que Probcons
MUSCLE	2004	Progressif	Très rapide, une bonne précision	Moins précis que Probcons
Probcons	2005	Progressif	Très précis	Moins rapide que MAFFT et MUSCLE
SAGA	1996	Itératif	Une précision acceptable	Très lent par rapport aux algorithmes progressifs

TAB. 4.2 – *Récapitulatif des algorithmes présentés.*

4.10 Les Benchmarks

Afin de déterminer la qualité d'un algorithme d'alignement multiple, nous avons besoin de comparer l'alignement produit par cet algorithme avec un alignement référence qui prend en compte le point de vue biologique. Pour cela, des jeux de tests ont été créés à partir de séquences réelles, et ils ont été alignés par des biologistes. Des bases de jeux d'essais (benchmarks) regroupent ces alignements, avec pour chaque jeu le meilleur résultat possible d'un point de vue biologique.

Actuellement, il existe plusieurs benchmaks d'alignements. Nous détaillerons plus particulièrement *BAliBASE*[1] que nous avons utilisé pour valider notre approche.

[1]BAliBASE : Benchmark Alignment dataBASE

4.10.1 *BAliBASE*

BAliBASE (Benchmark Alignment dataBASE) [60] est la première base des jeux d'essais pour les protéines qui a été massivement citée dans les publications. Même si d'autres bases existent maintenant, elle continue à être une référence incontournable. Dans sa première version, elle comportait 142 jeux de séquences, répartis en 5 catégories, appelées *références*. Chacune de ces références correspond à une classe différente de problèmes. Citons par exemple, le problème de la séquence orpheline (Référence 2), qui n'a aucune similarité avec les autres séquences. Aussi le problème des séquences de tailles très différentes, ou nécessitant de très grandes brèches (longueur supérieure à 100).

Actuellement il existe de nouvelles versions *BAliBASE 2.0* [61] et *BAliBASE 3.0* [62]. Elles sont basées sur la version 1. Elles reprennent les 5 références existantes avec quelques modifications dans les résultats. Elles incluent en plus des nouvelles références pour divers problèmes, dans la version 3 le nombre de séquences est devenu 6255.

Remarque : Les tests que l'on trouve dans la littérature sont réalisés juste sur les 5 premières références.

Chaque jeu d'essais de *BAliBASE* est proposé avec la meilleure solution. Cela ne permet toutefois pas de pouvoir réaliser des comparaisons entre le résultat d'un algorithme et la solution optimale ou entre les résultats de plusieurs algorithmes.

Pour cela deux fonctions de comparaison ont également été ajoutées, chacune permettant de montrer un critère de qualité pour les alignements. La première *SPS* correspond à un critère de comparaison local, la deuxième *CS* réalise une comparaison plus globale de l'alignement.

La fonction de comparaison *SPS* (*Sum-of-Pairs Score*) est basée sur le principe de la fonction de somme des paires. Toutes les paires de séquences sont parcourues, aussi bien dans l'alignement de référence que dans l'alignement résultat. Pour chacun des deux alignements, les paires de résidus identiques ont la valeur 1, et les autres ont la valeur 0. Cette méthode consiste donc à déterminer le nombre de paires de résidus identiques entre la référence et le résultat (Figure 4.10). En divisant cette somme par le nombre total de paires de résidus de la référence, on obtient un pourcentage de similarité entre les paires de résidus des deux alignements.

La fonction de comparaison *CS* (Column Score) est quant à elle basée sur un point de vue différent du concept d'alignement. La qualité que l'on attribue dans ce cas à un alignement

FIG. 4.10 – *Principe du critère SPS.*

multiple dépend d'une colonne complète bien alignée. Réussir à obtenir une paire de résidus identique entre la référence et le résultat ne suffit plus, il faut obtenir l'identité entre tous les résidus d'une même colonne (Figure 4.11). Le critère de comparaison *CS* se calcule en faisant la somme de toutes les colonnes identiques entre l'alignement de référence et l'alignement résultat. Pour obtenir un pourcentage, ce nombre est divisé par le nombre de colonnes de l'alignement de référence.

FIG. 4.11 – *Principe du critère CS.*

BAliBASE propose un programme écrit en langage C (bali_score.c) qui permet d'évaluer les valeurs *SPS* et *CS*.

4.10.2 Les autres bases

Comme nous l'avons déjà mentionné, il existe actuellement plusieurs benchmarks d'alignement multiple de séquences, nous citons ici ceux qui sont les plus utilisées [63] :

- *HOMSTRAD* [64],
- *Oxbench* [65],
- *Prefab* [52],
- *SABmark* [66],

– *IRMBASE* [67].

4.11 Conclusion

L'alignement multiple de séquences est une problématique très importantes de la bioinformatique. En effet aligner des séquences constitue un problème à part entière, mais il est également utilisé comme point de départ pour d'autres problèmes de bioinformatique.

Dans ce chapitre nous avons défini l'alignement multiple de séquences et donner son utilisation dans le domaine de la bioinformatique. Comme pour l'alignement par paires, nous avons besoin d'une fonction d'évaluation qui détermine la qualité du résultat obtenu. Pour cela nous avons présenté quelques fonctions d'évaluations les plus utilisées.

Il existe un nombre très important d'algorithmes qui traitent le problème d'alignement multiple, ils peuvent être classifiés en trois grandes catégories selon le principe général qu'ils utilisent.

- Les algorithmes basés sur une méthode exacte sont minoritaires car ils ne peuvent être utilisés que pour des alignements ne comportant que peu de séquences.
- Les algorithmes progressifs qui sont basés sur l'algorithme d'alignement de deux séquences. Ils commencent par construire un arbre qui sert à déterminer l'ordre dans lequel les séquences doivent être alignées, puis ils alignent progressivement les séquences.
- Les algorithmes itératifs utilisant des méthodes plus variées, ils essayent d'aligner toutes les séquences simultanément.

Nous avons essayé par la suite dans ce chapitre de faire une petite comparaison entre les algorithmes présentés. nous avons donné la classe de chaque algorithme, son principal avantage et inconvénient.

Pour pouvoir comparer les résultats obtenus des différents algorithmes, des benchmarks ont été créés. Ils contiennent en général des ensembles de séquences alignés par des biologistes, ces ensembles sont appelés des références. Plus que le résultat fournis par un algorithme est proche de la référence, plus qu'il est de bonne qualité.

Chapitre 5

Une approche hybride bio-inspirée pour la résolution du problème d'alignement multiple de séquences

5.1 Introduction

Nous avons vu au chapitre précédent que le problème d'alignement multiple de séquences a été démontré NP-Complet. Aucune méthode utilisée jusqu'à présent ne peut résoudre le problème efficacement dans tous les cas. Il existe plusieurs algorithmes pour la résolution du problème d'alignement multiple de séquences, ils peuvent être classés en trois catégories :

- Les algorithmes exacts : ils ne peuvent être utilisés que pour un nombre très limité de séquences à cause de la quantité de mémoire nécessaire,
- Les algorithmes progressifs : ils sont les plus utilisés car ils donnent en général des résultats acceptables dans un temps limité,
- Les algorithmes itératifs : ils nécessitent en général des temps de calculs très importants.

Nous proposons dans ce mémoire une approche hybride bioinspirée que nous appelons *MMGA* (*Muscle Mafft Genetic Algorithm*). L'approche proposée tire profit de la convergence rapide (entraînant une exécution rapide) des algorithmes progressifs en combinant les résultats de deux algorithmes réputés par leur rapidité (*MAFFT* et *MUSCLE*) à l'aide d'un algorithme génétique, ce qui permet d'accroître la précision des résultats tout en préservant un temps d'exécution réduit.

5.2 Présentation de l'approche proposée

Comme nous l'avons déjà signalé au chapitre précédent, les algorithmes progressifs commencent par aligner des sous-groupes de séquences puis essayent de les combiner entre eux pour former des alignements contenant de plus en plus de séquences. Cette stratégie peut influencer une perte d'informations car traiter des séquences deux à deux est moins précis que de les traiter toutes ensemble. Pour cette raison certains algorithmes progressifs proposent une phase de raffinement itérative afin d'améliorer les résultats.

Notre approche combine trois algorithmes : *MUSCLE* et *MAFFT* et un algorithme génétique. *MUSCLE* [52] (cf. 4.7.4) et *MAFFT* [51] (cf. 4.7.3) sont des algorithmes d'alignements multiples de séquences très connus. Ils sont réputés par leur rapidité par rapport aux autres méthodes, et ils donnent en générale de bons résultats.

Le troisième algorithme est un algorithme de type génétique, il a le rôle d'améliorer les alignements obtenus par les deux premiers algorithmes. Cet algorithme doit évaluer la qualité de chaque alignement, les combiner à l'aide des différents opérateurs pour produire de nouveaux alignements et donner le meilleur résultat. La figure 5.1 illustre le schéma général de l'approche proposée.

5.3 Présentation de l'algorithme génétique utilisé

5.3.1 Génération de la population initiale

Contrairement à *SAGA* [56] (cf. 4.8.1) qui génère la population initiale par insertion aléatoire de brèches, notre algorithme utilise les résultats des alignements obtenus à partir de deux algorithmes (*MUSCLE* et *MAFFT*) reconnus par leur vitesse et donnant généralement de bons résultats.

Les deux alignements parents doivent être au format standard (*Fasta format*[1]). Une séquence au *format FASTA* commence par une ligne de titre (*nom, définition* ...), suivie par les lignes de la séquence. La ligne de titre se distingue de la séquence par un symbole plus grand que (">") au début de ligne. Il est recommandé que toutes les lignes de texte ne dépassent 80 caractères de longueur. Voici un exemple d'une séquence au *format Fasta* (figure 5.2) :

[1]Format standard de représentation des séquences

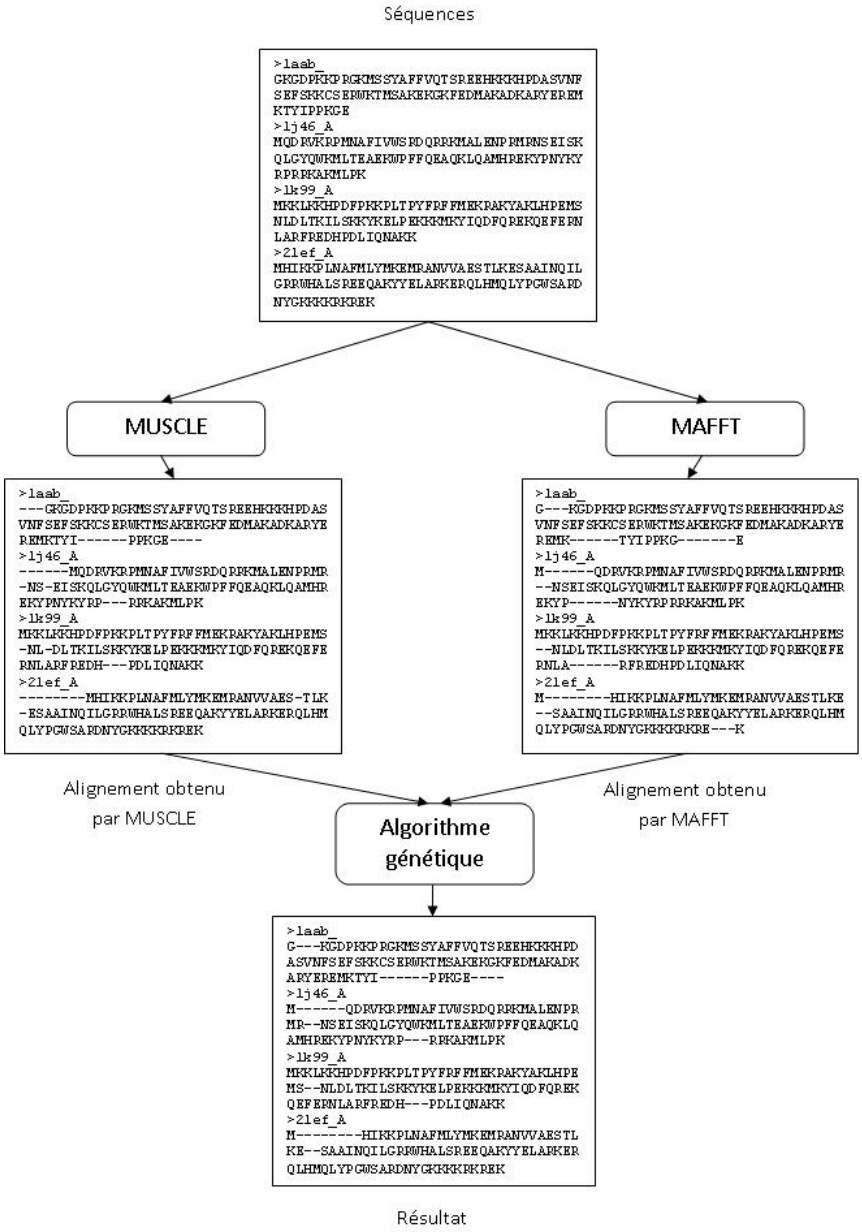

FIG. 5.1 – *Schéma général de l'approche proposée.*

```
>PIG1_BOVIN
MAGAASPCANGCGPSAPSDAEVVHLCRSLEVGTVMTLFYSKKSQRPERKTFQVKLETRQI
TWSRGADKIEGAI----------------------------------------------------
----------------------------------------------------------------
-----------------DIREIKEIRPGKTSRDFDRYQEDPAFRPDQSHCFVILYGMEF
RLKTLSLQATSEDEVNMWIRGLTWLMEDTLQAATPLQIERWLRKQFYSVDRNREDRISAK
DLKNMLSQVNYRVPNMRFLRERLTDLEQRTSDITYGQFAQLYRSLMYSAQKTMDLPFLEA
SALRAGERPELCRVSLPEFQQFLLEYQGELWAVDRLQVQEFMLSFLRDPLREIEEPYFFL
DEFVTFLFSKENSIWNSQLDEVCPDTMNNPLSHYWISSSHNTYLTGDQFSSESSLEAYAR
CLRMGCRCIELDCWDGPDGMPVIYHGHTLTTKIKFSDVLHTIKEHAFVASEYPVILSIED
HCSIAQQRNMAQYFKKVLGDTLLTKPVDIAADGLPSPNQLKRKILIKHKKLAEGSAYEEV
PTSVMYSENDIS----------------------------NSIKNGILYLEDPVNHEWYP
```

FIG. 5.2 – *Séquence en FASTA format.*

5.3.2 La fonction d'évaluation

La fonction d'évaluation que nous utilisons pour déterminer la qualité d'un alignement est la fonction de *somme des paires* (*SP*) (cf. 4.4.1). Pour déterminer le coût de chaque paire de résidus, nous utilisons la matrice *Blosum 62* [29] (cf. 3.3.5.2).

Le calcul du coût des brèches est effectué suivant le modèle de *coût affine* (cf. 3.3.6.2), avec comme paramètre par défaut le coût d'ouverture d'une brèche est $gop = -10$, et le coût de l'extension de la brèche $gep = -1$. Ces paramètres peuvent être changés selon le type des séquences en entrées. Par exemple si les séquences initiales sont de tailles très différentes, l'alignement résultat va contenir de très grandes brèches donc il serait avantageux de ne pas trop pénaliser les extensions des brèches, en leur affectant un coût ($gep = -0.1$).

5.3.3 Le Croisement

Le croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Il consiste à combiner deux individus (parents) pour générer deux individus (enfants ou descendants) héritant des caractéristiques de leurs parents (voir la figure 5.3).

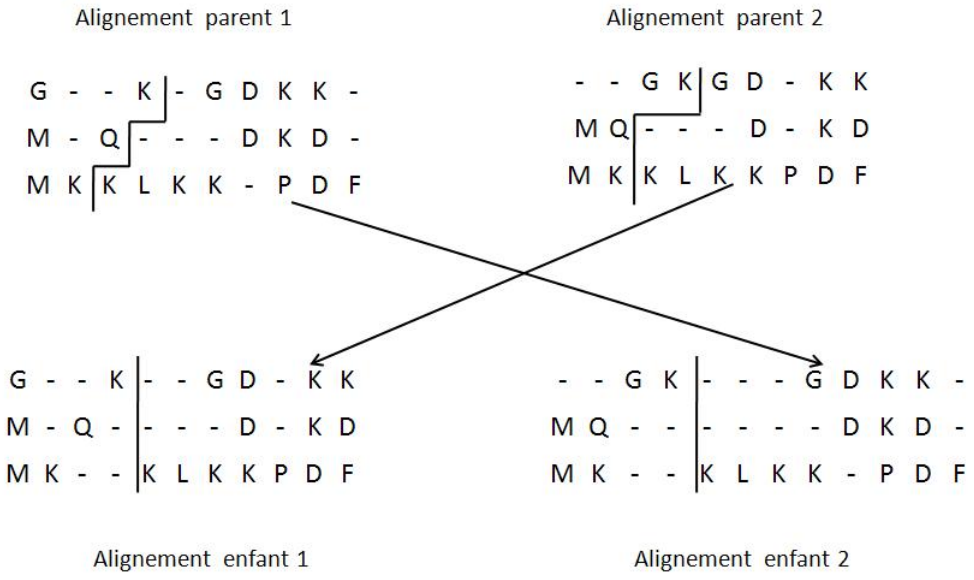

FIG. 5.3 – *Exemple d'un croisement .*

Dans notre cas, nous prenons deux alignements (parents) qu'on coupe en deux parties. La position du point de découpage est un paramètre important dans la convergence vers

l'alignement optimal. Nous changeons cette position 5 fois afin d'obtenir 10 différents descendants.

5.3.4 La sélection

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais. Dans notre approche, nous évaluons les alignements descendants obtenus à l'aide de la fonction d'évaluation (*SP*), puis nous choisissons les trois meilleurs alignements afin d'être utilisés pour produire d'autres générations. Cette méthode de sélection directe des meilleurs individus a l'avantage de permettre une convergence rapide des solutions.

5.3.5 Algorithme d'alignement multiple *MMGA*

Après avoir défini les différents paramètres de notre algorithme génétique, nous présentons les étapes de l'algorithme *MMGA* :

Algorithme MMGA

1. **Générer la population initiale** en utilisant *MUSCLE* et *MAFFT*
2. **Évaluer** les deux alignements parents
3. Faire le **croisement** et générer 10 descendants
4. **Évaluer** les descendants
5. **Sélectionner** les trois meilleurs descendants
6. **Sauvegarder** les trois meilleurs descendants (*d1, d2, d3*) et abandonner le reste
7. **Si** (la population est stabilisée) **Ou** (condition d'arrêt) **alors** Aller à **9**
8. Aller à **3** et faire le **croisement** des meilleurs descendants (*d1 d2, d1 d3, d2 d3*)
9. **Afficher** le meilleur alignement
10. **Fin**

5.4 Les données de test

Afin de valider notre approche, nous avons effectué des tests sur 63 ensembles de séquences protéiques à partir de différents références de *BAliBASE 3.0* [62] (cf. 4.10.1). Le nombre de séquences contenues dans chaque ensemble et les tailles des séquences sont très variés ce qui permet de bien tester notre approche.

Nous présentons dans le tableau suivant les données de tests utilisées en donnant le nom de chaque ensemble, le nombre de séquences, la taille de la plus petite séquence, la taille de la plus grande séquence et une petite description de chaque ensemble de tests.

l'ensemble de tests	**Nombre de séquences**	**Taille de la plus petite séquence**	**Taille de la plus grande séquence**	**Description**
BB11001	4	83	93	SHORT high mobility group protein
BB11002	8	52	193	SHORT SH3
BB11003	4	414	516	LONG aldehyde dehydrogenase
BB11004	4	390	456	LONG histidyl-trna synthetase
BB11005	14	329	465	LONG aminotransferase
BB11006	8	186	283	MEDIUM foot-and-mouth disease virus
BB11007	9	385	457	LONG cytochrome p450
BB11008	4	104	540	SHORT SH2
BB11009	4	97	337	SHORT ferredoxin [2fe-2s]
BB11010	4	490	492	MEDIUM glucosamine 6-phosphate synthase
BB11011	5	160	242	MEDIUM hepatitis proteinase
BB11012	4	320	397	LONG serpins
BB11013	5	51	101	SHORT myb dna-binding domain
BB11014	6	502	634	LONG acetyl-coa synthetase
BB11015	4	297	327	MEDIUM lactate dehydrogenase

BB11016	8	316	729	LONG dihydrolipoamide dehydrogenase
BB11017	4	247	264	MEDIUM alpha tricosanthin
BB11018	14	418	750	LONG cyclodextrin
BB11019	10	299	396	LONG alcohol dehydrogenase
BB11020	9	201	237	MEDIUM glutathione
BB11021	4	102	139	SHORT plastocyanin
BB11022	4	63	205	SHORT repressor
BB11023	7	231	407	MEDIUM sulfate binding protein
BB11024	4	372	465	LONG seryl-tRNA synthetase
BB11025	4	64	103	SHORT pertussis toxin
BB11026	7	76	906	SHORT ubiquitin
BB11027	7	175	432	MEDIUM uridylate kinase
BB11028	10	93	211	SHORT twitchin
BB11029	4	81	138	SHORT cytochrome
BB11030	14	236	392	MEDIUM 3-Alpha, 20 beta-hydroxysteroid dehydrogenase
BB11031	11	300	611	LONG myrosinase
BB11032	8	226	403	MEDIUM phtalate reductase
BB11033	11	85	239	SHORT thioredoxin
BB11034	8	401	729	MEDIUM glutathione reductase
BB11035	5	71	138	SHORT cytochrome c
BB11036	8	298	436	LONG enolase
BB11037	5	335	1192	LONG gal4
BB11038	8	261	614	MEDIUM protein kinase
BB20001	16	74	697	high mobility group protein
BB20002	20	52	1520	SH3
BB20003	74	409	800	aldehyde dehydrogenase
BB20004	55	401	734	histidyl-trna synthetase
BB30001	116	228	800	aldehyde dehydrogenase
BB30002	31	401	742	histidyl-trna synthetase

BB30003	142	329	514	aminotransferase
BB30004	50	378	748	acyl coa dehydrogenase
BB30005	113	284	1061	cytochrome p450
BB30006	18	97	923	SH2
BB30010	50	503	1293	acetyl-coa synthetase
BB40001	28	72	1403	high mobility group protein
BB40003	12	421	1002	carboxypeptidase
BB40004	67	454	1320	aldehyde dehydrogenase
BB40005	12	401	734	histidyl-trna synthetase
BB40006	14	354	592	aminotransferase
BB40009	19	367	1066	cytochrome p450
BB40010	9	67	214	cold shock protein
BB50001	34	390	667	histidyl-trna synthetase
BB50002	13	172	819	eftu
BB50003	43	238	1098	glucosamine 6-phosphate synthase
BB50004	9	386	505	phosphoglycerate kinase
BB50005	11	642	844	lactoferrin
BB50009	28	265	553	glycyl-tRNA synthetase
BB50010	17	372	688	seryl-tRNA synthetase

5.5 Résultats et évaluation

Le but de notre travail est de proposer une méthode qui permet d'accroître la précision et d'améliorer la qualité des alignements multiples obtenus. Nous avons évalué la qualité des alignements multiples obtenus à l'aide des deux critères de comparaison proposés par *BAliBASE* qui sont *Sum-of-pairs Score* (*SPS*) et *Column Score* (*CS*) (cf. 4.10.1). Les deux critères permettent de donner un degré de similitude entre l'alignement résultat et l'alignement référence.

5.5.1 Comparaison avec *MUSCLE* et *MAFFT*

Les premiers tests sont faits sur les trois algorithmes utilisés dans notre approche. Nous avons évalué la qualité des alignements multiples obtenus par *MUSCLE*, *MAFFT* et *MMGA*. Les résultats sont prometteurs dans la mesure où on se trouve dans l'un des deux cas suivant dans la plupart des tests :

1. ***MMGA* marque un score meilleur que *MAFFT* et *MUSCLE* :**
   MMGA améliore le score *SPS* ou *CS* ou les deux en même temps comme le montre l'exemple du test effectué sur *BB110001* (figure 5.4).

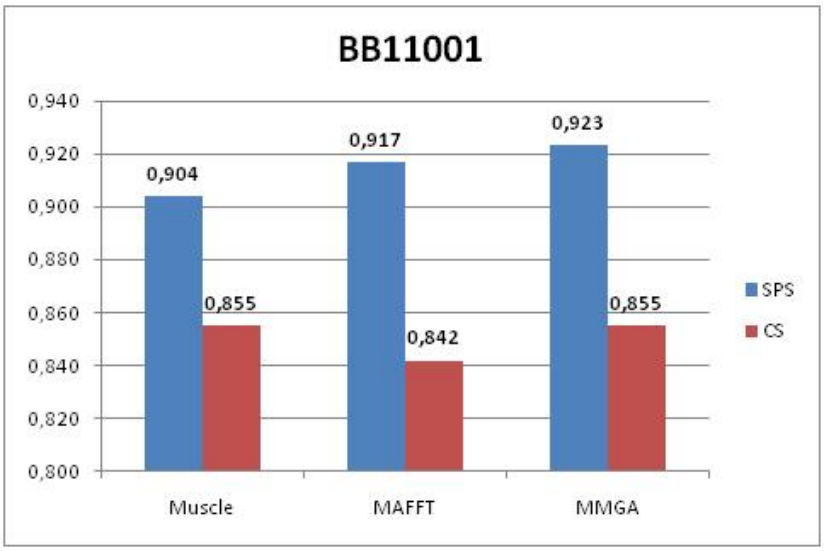

FIG. 5.4 – *Histogramme des Scores SPS et CS dans le test BB11001.*

On peut remarquer que l'alignement obtenu par l'algorithme *MMGA* est meilleur, il donne un score de $SPS = 0,923$ et $CS = 0,855$.

2. ***MMGA* donne un score égal au meilleur score entre *MAFFT* et *MUSCLE* :**
   Ce cas est le plus fréquent dans le processus d'alignement. Dans certains tests *MAFFT* donne de meilleurs scores que *MUSCLE*, ce qui implique que *MMGA* produit un alignement similaire à celui de *MAFFT* comme le montre l'exemple du test effectué sur *BB11009* (figure 5.5).

Dans d'autre test *MUSCLE* donne un meilleur score que *MAFFT* comme le montre

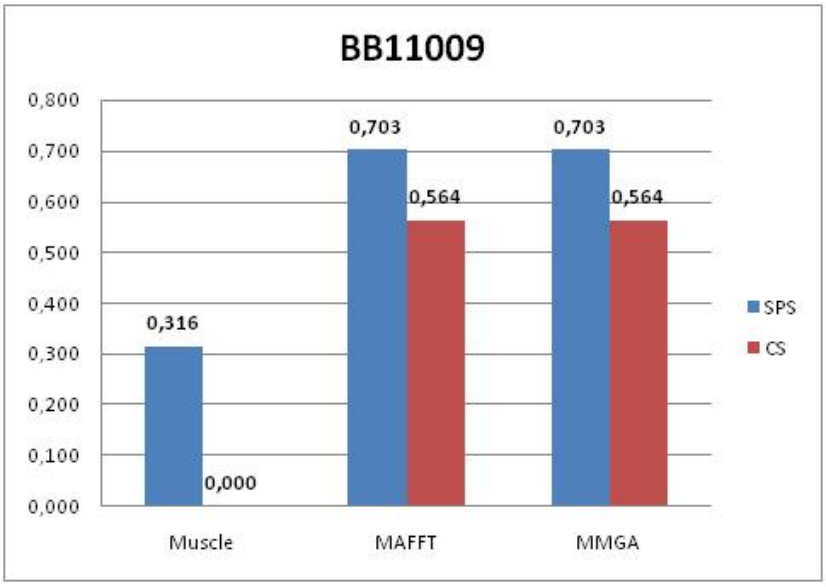

FIG. 5.5 – *Histogramme des Scores SPS et CS dans le test BB11009.*

l'exemple du test effectué sur *BB11035* (figure 5.6).

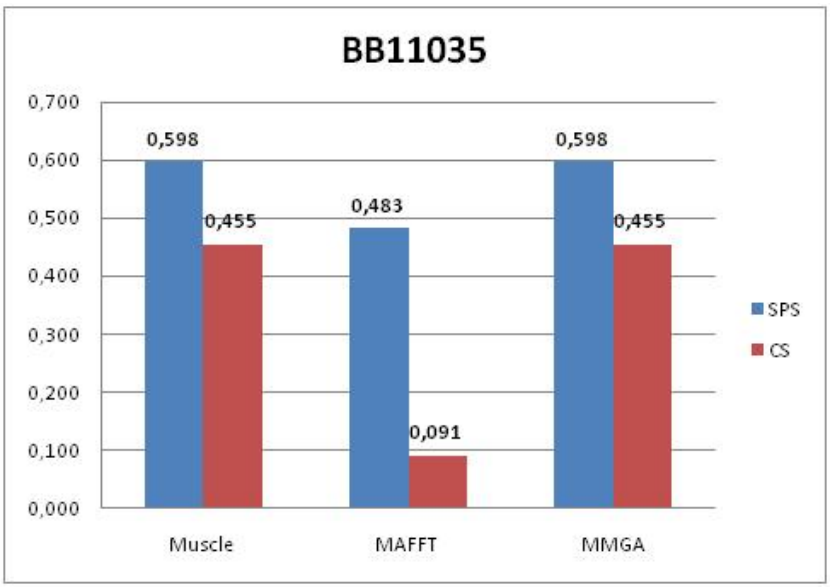

FIG. 5.6 – *Histogramme des Scores SPS et CS dans le test BB11035.*

Les différents tests effectués montrent que notre algorithme produit des alignements meilleurs que ceux obtenus avec *MUSCLE* et *MAFFT*. En effet, au pire il donnera un alignement similaire au meilleur alignement parents dans le cas où aucun descendants n'améliore le résultat.

5.5.2 Comparaison avec les autres méthodes

Afin de valider notre approche nous l'avons comparé avec d'autres algorithmes d'alignement multiple de séquences. En plus de *MUSCLE* et *MAFFT*, nous avons effectué des tests sur les résultats de *Clsutal W* [47] (cf. 4.7.1) et *ProbCons* [55] (cf. 4.7.5).

Clustal W est un programme d'alignement multiple très populaire. A la fin des années 1990, *Clustal W* était le programme d'alignement multiple le plus utilisé [48], et il reste encore aujourd'hui très utilisé.

ProbCons est actuellement l'algorithme d'alignement multiple de séquences le plus précis [1] [6]. Il réalise statistiquement une amélioration significative par rapport à d'autres méthodes tout en préservant une vitesse pratique[2] [55].

Les tests ont été effectués sur les 63 cas de *BAliBASE 3.0* décrits précédemment, et les versions des programmes utilisés sont :

- MUSCLE 3.70 + fix 1-2
- MAFFT 6.717-1
- ProbCons 1.12-4
- Clustal W 2.0.10-1

Tous ces programmes ont été utilisés avec les paramètres par défaut (automatique). Dans les quatre programmes le paramètre par défaut favorise la précision des résultats et non pas la vitesse d'exécution.

Le tableau suivant contient tous les tests effectués. Nous donnons les scores *SPS* et *CS* enregistrés par chaque algorithme où nous mettons en gras souligné les cas ou *MMGA* marque un meilleur score par rapport aux autres algorithmes.

Test	MUSCLE		MAFFT		Clustal W		ProbCons		MMGA	
	SPS	CS	SPS	CS	SPS	CS	SPS	CS	SPS	CS
BB11001	0,904	0,855	0,917	0,842	0,961	0,921	0,939	0,895	0,923	0,855
BB11002	0,721	0,489	0,517	0,000	0,453	0,000	0,527	0,000	**0,721**	**0,489**
BB11003	0,547	0,393	0,653	0,464	0,609	0,430	0,662	0,483	0,653	0,464
BB11004	0,598	0,428	0,526	0,323	0,196	0,000	0,623	0,434	0,597	0,431
BB11005	0,411	0,088	0,430	0,088	0,371	0,101	0,397	0,104	0,411	0,088

[2]Métrique propre au spécialistes biologistes en fonction de l'opération effectuée : recherche de motif dans une base, alignement de deux ou plusieurs séquences, etc.

BB11006	0,459	0,267	0,407	0,247	0,250	0,000	0,448	0,253	**0,459**	**0,267**
BB11007	0,608	0,402	0,693	0,497	0,557	0,263	0,697	0,447	0,693	**0,497**
BB11008	0,393	0,233	0,583	0,444	0,363	0,000	0,546	0,422	**0,583**	**0,444**
BB11009	0,316	0,000	0,703	0,564	0,346	0,000	0,690	0,590	**0,703**	0,564
BB11010	0,236	0,065	0,317	0,139	0,201	0,000	0,344	0,168	0,317	0,139
BB11011	0,152	0,000	0,270	0,188	0,198	0,070	0,341	0,141	0,270	**0,188**
BB11012	0,869	0,779	0,858	0,792	0,910	0,859	0,924	0,879	0,869	0,779
BB11013	0,053	0,000	0,235	0,000	0,084	0,000	0,155	0,000	**0,235**	0,000
BB11014	0,753	0,581	0,767	0,608	0,707	0,535	0,792	0,650	0,767	0,608
BB11015	0,827	0,756	0,705	0,538	0,654	0,437	0,707	0,552	**0,827**	**0,756**
BB11016	0,382	0,000	0,623	0,272	0,367	0,000	0,473	0,000	**0,623**	**0,272**
BB11017	0,732	0,619	0,780	0,678	0,653	0,485	0,673	0,540	0,750	0,634
BB11018	0,483	0,240	0,626	0,293	0,432	0,156	0,593	0,305	**0,626**	0,293
BB11019	0,625	0,081	0,656	0,195	0,495	0,074	0,644	0,148	**0,656**	**0,195**
BB11020	0,589	0,292	0,692	0,316	0,652	0,327	0,681	0,322	**0,692**	0,316
BB11021	0,562	0,450	0,655	0,475	0,173	0,000	0,702	0,562	0,655	0,475
BB11022	0,073	0,000	0,096	0,000	0,251	0,000	0,137	0,000	0,096	0,000
BB11023	0,455	0,127	0,524	0,249	0,373	0,156	0,556	0,307	0,524	0,249
BB11024	0,197	0,000	0,417	0,237	0,173	0,000	0,462	0,193	0,417	**0,237**
BB11025	0,095	0,000	0,042	0,000	0,118	0,000	0,154	0,000	0,095	0,000
BB11026	0,305	0,000	0,211	0,000	0,250	0,000	0,291	0,000	**0,305**	0,000
BB11027	0,368	0,172	0,391	0,172	0,253	0,000	0,441	0,172	0,368	**0,172**
BB11028	0,411	0,000	0,434	0,000	0,465	0,000	0,438	0,000	0,434	0,000
BB11029	0,432	0,333	0,437	0,288	0,505	0,470	0,513	0,439	0,455	0,303
BB11030	0,373	0,102	0,576	0,305	0,330	0,112	0,614	0,203	0,576	**0,305**
BB11031	0,427	0,000	0,543	0,226	0,278	0,000	0,521	0,117	**0,543**	**0,226**
BB11032	0,598	0,311	0,732	0,399	0,520	0,166	0,751	0,435	0,732	0,399
BB11033	0,418	0,000	0,473	0,133	0,443	0,000	0,404	0,108	**0,473**	**0,133**
BB11034	0,419	0,000	0,618	0,319	0,277	0,000	0,485	0,000	**0,618**	**0,319**
BB11035	0,598	0,455	0,483	0,091	0,468	0,303	0,515	0,348	**0,598**	**0,455**
BB11036	0,550	0,140	0,595	0,279	0,480	0,131	0,548	0,231	**0,595**	**0,279**

BB11037	0,412	0,187	0,437	0,216	0,364	0,117	0,529	0,290	0,437	0,216
BB11038	0,653	0,361	0,752	0,526	0,485	0,000	0,821	0,604	0,752	0,526
BB20001	0,477	0,000	0,496	0,000	0,424	0,000	0,456	0,000	**0,496**	0,000
BB20002	0,223	0,000	0,674	0,000	0,152	0,000	0,668	0,000	**0,674**	0,000
BB20003	0,947	0,263	0,950	0,298	0,934	0,239	0,944	0,283	**0,950**	**0,298**
BB20004	0,850	0,582	0,854	0,553	0,825	0,000	0,852	0,587	**0,854**	0,553
BB30001	0,753	0,111	0,818	0,088	0,697	0,124	0,831	0,160	0,753	0,111
BB30002	0,756	0,348	0,782	0,376	0,517	0,000	0,781	0,368	**0,782**	**0,376**
BB30003	0,554	0,013	0,527	0,063	0,452	0,032	0,561	0,047	0,554	0,013
BB30004	0,810	0,484	0,870	0,614	0,817	0,505	0,850	0,579	**0,870**	**0,614**
BB30005	0,797	0,283	0,771	0,251	0,703	0,289	0,794	0,310	**0,797**	0,283
BB30006	0,509	0,000	0,724	0,402	0,604	0,261	0,680	0,228	**0,724**	**0,402**
BB30010	0,865	0,437	0,872	0,511	0,841	0,277	0,869	0,515	**0,872**	0,511
BB40001	0,781	0,000	0,896	0,776	0,779	0,000	0,886	0,718	**0,896**	**0,776**
BB40003	0,894	0,794	0,865	0,702	0,827	0,652	0,910	0,780	0,894	0,749
BB40004	0,903	0,453	0,903	0,468	0,864	0,417	0,905	0,393	0,903	**0,468**
BB40005	0,874	0,687	0,876	0,702	0,874	0,715	0,879	0,726	0,876	0,702
BB40006	0,733	0,267	0,766	0,350	0,696	0,300	0,737	0,312	**0,766**	**0,350**
BB40009	0,715	0,404	0,742	0,399	0,731	0,407	0,739	0,410	**0,742**	0,399
BB40010	0,820	0,530	0,828	0,470	0,817	0,530	0,853	0,561	0,828	0,470
BB50001	0,747	0,390	0,767	0,346	0,609	0,237	0,761	0,380	0,747	**0,390**
BB50002	0,308	0,000	0,448	0,000	0,232	0,000	0,398	0,000	**0,448**	0,000
BB50003	0,586	0,295	0,621	0,355	0,424	0,021	0,601	0,337	**0,621**	0,355
BB50004	0,952	0,860	0,962	0,887	0,945	0,813	0,956	0,862	0954	0,860
BB50005	0,914	0,705	0,916	0,679	0,917	0,702	0,930	0,741	0,914	0,705
BB50009	0.660	0,000	0,723	0,240	0,659	0,000	0,722	0,189	**0,723**	**0,4240**
BB50010	0,509	0,000	0,760	0,333	0,462	0,000	0,759	0,314	**0,760**	**0,333**

D'après le tableau, dans 39 cas sur 63, *MMGA* donne un score supérieur ou égal au meilleur score marqué par les autres algorithmes.

Afin de bien éclaircir les résultats nous avons calculé la moyenne des scores *SPS* et *CS* pour chaque algorithme. Les moyennes sont données dans le tableau 5.3 :

Score / Algorithme	SPS	CS
MUSCLE	0,570	0,271
MAFFT	0,632	0,338
Clustal W	0,515	0,201
ProbCons	0,636	0,336
MMGA	**0,643**	**0,358**

TAB. 5.3 – *Moyennes des scores SPS et CS pour chaque algorithme .*

L'histogramme des moyennes des scores *SPS* est donnée dans la figure 5.7 :

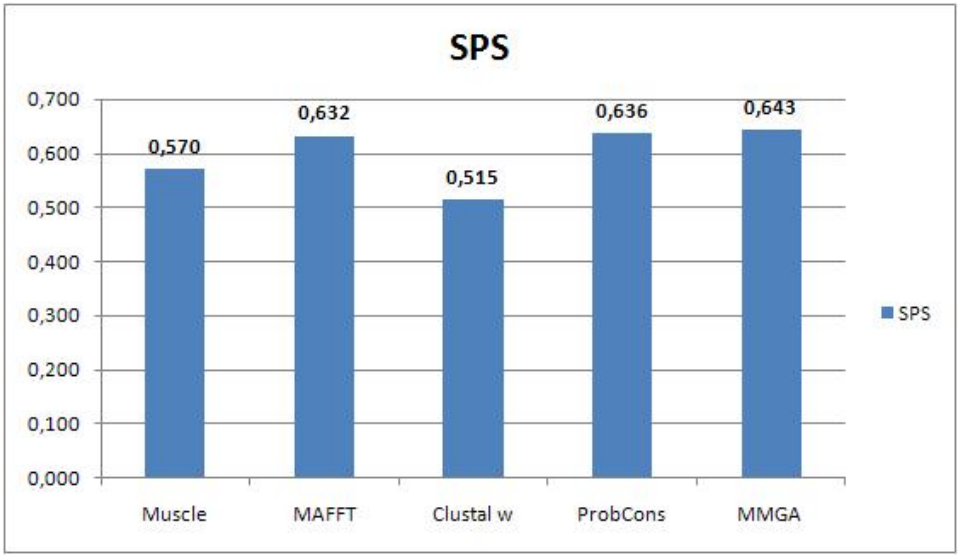

FIG. 5.7 – *Histogramme des moyennes des Scores SPS de chaque algorithme.*

D'après la figure 5.7 l'algorithme *ProbCons* marque un score meilleur que celui de *MUSCLE*, *MAFFT* et *Clustal W* mais il est moins performant par rapport à notre approche *MMGA*. *MMGA* donne le meilleur score avec $SPS = 0,643$.

La figure 5.8 contient l'histogramme des moyennes des scores *CS*.

Comme pour *SPS*, notre approche *MMGA* donne un score $CS = 0,358$ qui est largement meilleur des scores donnés par les autres algorithmes.

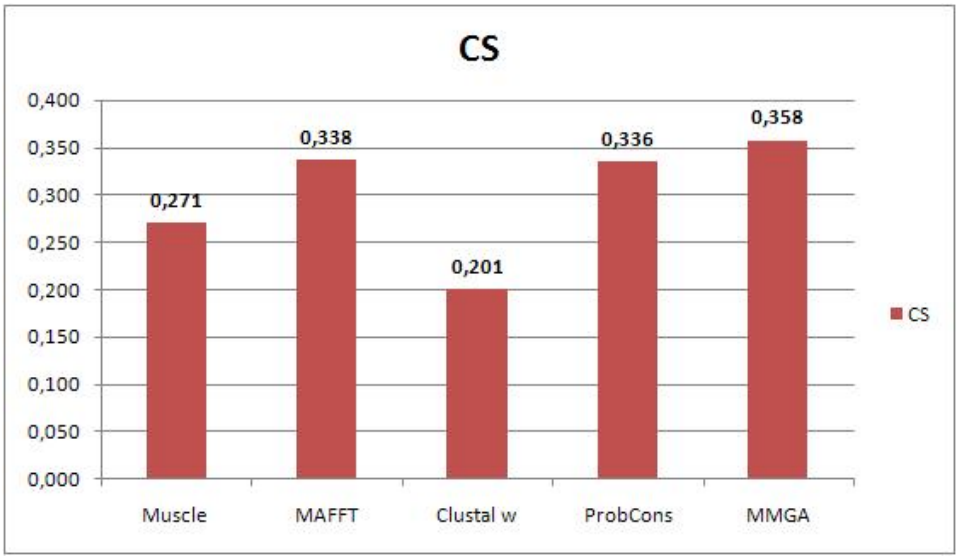

FIG. 5.8 – *Histogramme des moyennes des Scores CS de chaque algorithme.*

Nous donnons dans l'histogramme suivant (figure 5.9) la moyenne des deux scores *SPS* et *CS* pour chauqe algorithme afin de voir le cumule des améliorations obtenus avec notre approche.

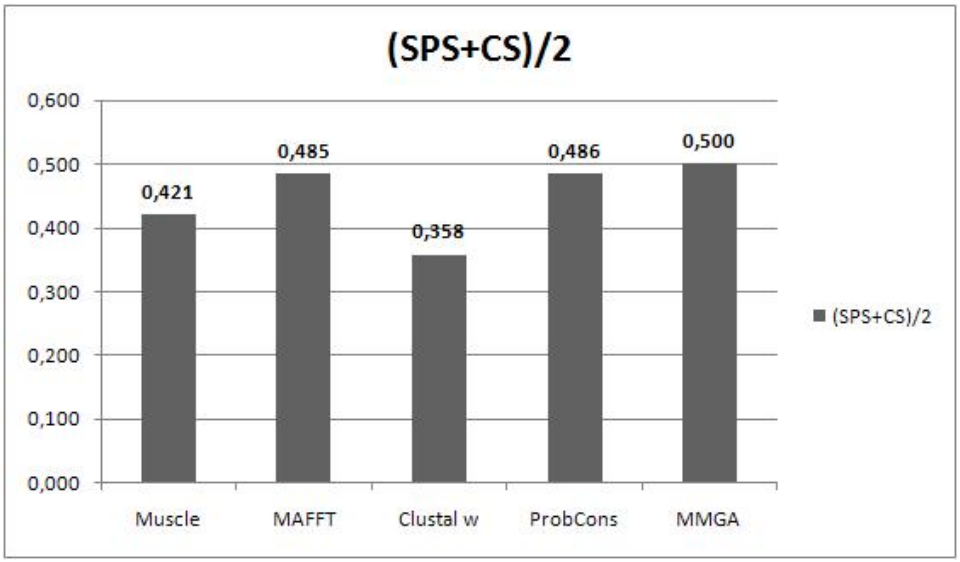

FIG. 5.9 – *Histogramme des sommes des moyennes des Scores SPS et CS.*

D'après l'histogramme de la figure 5.9, *MMGA* dépasse de loin les autres algorithmes et il marque un score de 0,500, alors que *ProbCons*, qui est considéré le plus précis par rapport aux autres méthodes, donne un score de 0,486. Ceci prouve que *MMGA* est la méthode la

plus précise.

5.5.3 Les temps d'exécutions

L'objectif de notre travail est de développer un algorithme permettant d'améliorer la qualité des résultats d'alignements. Cet objectif est atteint comme nous l'avons déjà présenté dans la section précédente. Pour les temps d'exécutions nous avons calculé le temps pris par chaque algorithme dans plusieurs cas. Nous présentons ici les tests effectués sur *BB20002* qui contient la plus longue séquence dans tous les tests (avec 1520 acides aminés) et *BB30003* contient le plus grand nombre de séquences (142 séquences). Ces deux tests donnent une idée générale sur les temps d'exécutions pris par chaque algorithme.

Pour faire les tests nous avons utilisé une machine dotée de :

- Un processeur Pentium(R) Dual-Core E5500 @ 2.80GHZ 2.80GHZ, 2,00 Go de RAM.
- Système linux(Ubuntu version 10.04 (lucid), Noyau Linux 2.6.32-25-generic, GNOME 2.30.2).

Dans le tableau 5.4 nous donnons les temps d'exécution pris par *MUSCLE*, *MAFFT* et l'algorithme génétique proposé (*GA*). *MMGA* contient la somme des temps d'exécutions des trois algorithmes précédents. *MMGAp* (pour *MMGA* parallèle) c'est le temps pris par notre méthode si *MUSCLE* et *MAFFT* sont lancés en parallèle. Le temps d'exécution de *MMGAp* est donné par la formule suivante :

$$T(MMGAp) = MAX(T(MUSCLE), T(MAFFT)) + T(GA)$$

Tel que : $T(X)$ est le temps d'exécution de l'algorithme X.
Le tableau 5.4 contient aussi les temps d'exécution de *Clustal W* et de *ProbCons*.

	MUSCLE	MAFFT	GA	MMGA	MMGAp	Clustal W	ProbCons
BB20002	22 s	17 s	3 s	42 s	25 s	16 s	54 s
BB30003	36 s	12 s	50 s	98 s	86 s	69 s	1226 s

TAB. 5.4 – *Temps d'exécutions de chaque algorithme dans les cas BB20002 et BB30003.*

L'histogramme de la figure 5.10, illustre les temps d'exécutions dans le cas *BB20002* :

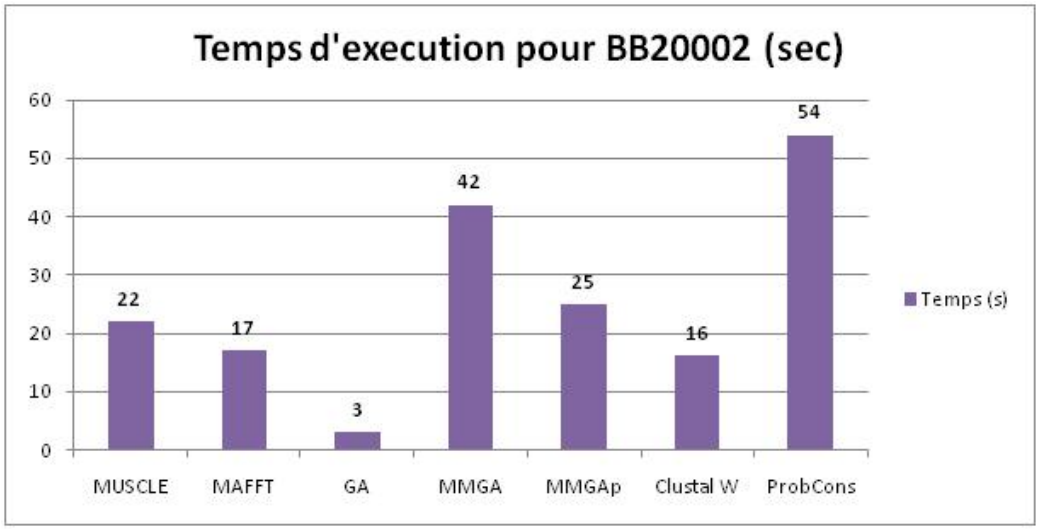

FIG. 5.10 – *Histogramme des temps d'exécutions de chaque algorithme dans le cas BB20002 (sec).*

ProbCons est l'algorithme le plus lent, il prend un temps d'exécution de 54 secondes. Même si *MMGA* est une hybridation de trois algorithmes lancés séquentiellement il prend 42 secondes, alors que *MMGAp* prend juste 25 secondes.

Le deuxième histogramme montre les temps d'exécution dans le cas *BB30003* (figure 5.11) :

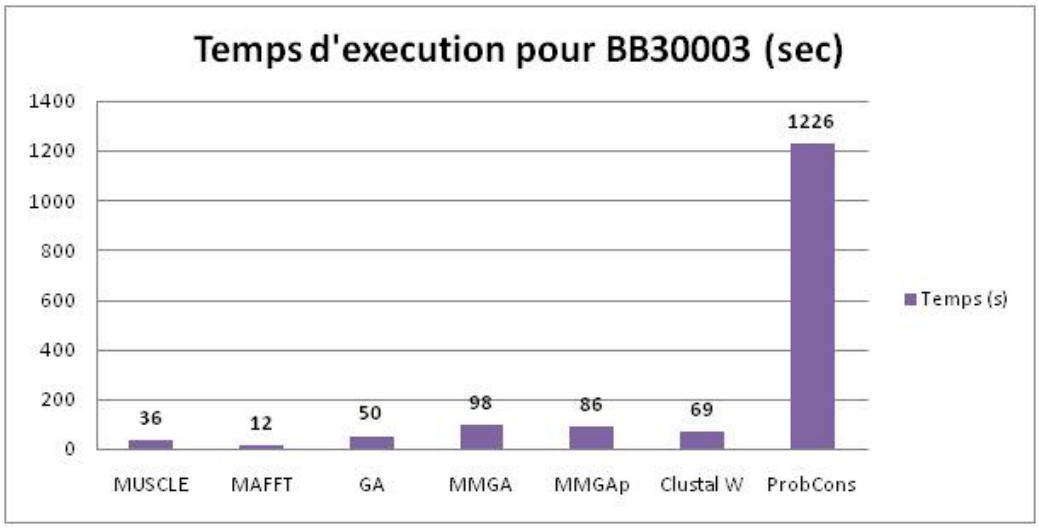

FIG. 5.11 – *Histogramme des temps d'exécutions de chaque algorithme dans le cas BB30003 (sec).*

Dans ce test, *ProbCons* reste plus de 20 minutes et 26 secondes, alors que *MMGA* ne dépasse pas une minute et 38 secondes. Pour *MMGAp* le temps pris est juste une minute

et 26 secondes. Dans les autres tests effectués toujours *ProbCons* est plus lent que *MMGA*. Tout cela prouve que notre approche est largement plus rapide que *ProbCons*.

5.6 Conclusion

Dans ce chapitre nous avons présenté une nouvelle approche hybride bioinspirée pour la résolution du problème d'alignement multiple de séquences baptisée *MMGA*. Elle combine trois algorithmes : *MUSCLE*, *MAFFT* et un algorithme génétique afin de produire des alignements de meilleure qualités , sinon notre approche permet de donner un alignement similaire au meilleur alignement en entrée. Cette dernière caractéristique est très importante car actuellement il existe un nombre très important de programmes d'alignement multiple de séquences, et chaque programme prétend de donner les meilleurs résultats.

Pour évaluer la qualité des alignements produit par notre algorithme nous avons utilisé un benchmark très populaire appelé *BAliBASE*(vresion 3.0). Nous avons comparé les résultats obtenues avec les résultats de quatre autres algorithmes d'alignement multiple de séquences considérés parmi les meilleurs qui sont *MUSCLE*, *MAFFT*, *Clustal W* et *ProbCons*. Les tests prouvent que notre approche donne, dans la plupart des cas, des alignements meilleurs que ceux donnés par les autres algorithmes tout en préservant des temps d'exécutions très petits par rapport au temps pris par l'algorithme *ProbCons*.

Un autre avantage de notre approche est qu'elle peut combiner des alignements multiples obtenus par n'importe quel programme d'alignements multiple de séquences, il suffit juste que les alignements soient en format standard (*FASTA Format*).

Conclusion générale

Au cours de ce travail de magistère, nous avons proposé une nouvelle approche hybride bioispirée pour la résolution du problème d'alignement multiple de séquences que nous avons appelé *MMGA*. Plusieurs méthodes ont été proposées pour la résolution de ce problème mais aucune ne peut le résoudre efficacement dans tous les cas. Les algorithmes *exacts* sont très gourmands en ressources et ils ne peuvent être utilisés que pour aligner un nombre très limité de séquences de petites tailles. Les algorithmes *progressifs* quand à eux sont très rapides et donnent en général de bons résultats, sauf qu'ils peuvent donner des résultats erronés à cause de la perte d'informations au cours du processus d'alignement progressif. Malgré que les algorithmes *itératifs* nécessitent en général des temps de calculs très importants et réalisent des alignements en prenant en compte toutes les séquences simultanément, leurs résultats sont moins précis que ceux des algorithmes progressifs dans la plupart des cas.

Notre approche combine trois algorithmes, deux algorithmes progressifs d'alignement multiple (*MAFFT* et *MUSCLE*) réputés d'être très rapides et un algorithme évolutionnaire de type génétique. Contrairement à *SAGA* (*Sequence Alignment by Genetic Algorithm*), un algorithme itératif d'alignement multiple réputé d'être gourmand en ressources et très lent, qui génère une population initiale de 100 individus par insertion aléatoire de brèches, notre algorithme utilise les alignements obtenus par *MAFFT* et *MUSCLE* ce qui permet une stabilisation de la population beaucoup plus rapide.

La fonction de fitness utilisée par notre algorithme est la *Somme des Paires* (*SP*). Nous avons utilisé comme matrice de substitution la matrice *Blosum 62* et le calcul du coût des brèches est effectué suivant le modèle de *coût affine* avec comme paramètre par défaut le coût d'ouverture d'une brèche est *gop = -10*, et le coût de l'extension de la brèche *gep = -1*. Avec cette matrice de substitution et ce modèle de calcul de coût des brèches, la fonction *Somme des Paire* donne en générale une bonne évaluation des alignements multiples dans un temps très petit.

Afin de produire de nouveaux descendants et d'enrichir la diversité de la population, notre algorithme applique différents opérateurs génétiques sur les alignements parents. Après cela les trois meilleurs descendants sont sélectionnés afin d'être utilisés pour produire d'autres générations. Ce processus est réitéré jusqu'à stabilisation de la population.

Nous avons testé notre approche sur différents ensembles de tests de *BAliBASE* (version 3.0). Cette dernière fournie avec chaque ensemble de tests un alignement référence qui prend en compte le point de vue biologique. Ensuite, nous avons comparé les résultats obtenus par notre approche avec ceux de différents algorithmes d'alignement multiple réputés d'être parmi les meilleurs qui sont : *MUSCLE*, *MAFFT*, *Clustal W* et *ProbCons*. Les résultats ont prouvé que notre approche donne des alignements meilleurs tout en préservant un temps de calcul très petit.

Comme perspectives, nous envisageons de combiner plusieurs algorithmes d'alignements multiples (plus de deux) ce qui permet d'accroître la précisions des résultats mais les temps de calculs seront beaucoup plus importants. Donc il sera recommandé d'utiliser cette approche dans le cas où le nombre de séquences est petit.

Nous pensons également à implémenter de nouvelles fonctions d'évaluations comme la *somme des paires pondérées* (*WSP*) ou bien la fonction *Coffee*, afin de comparer la qualité des alignements obtenus.

Dans les algorithmes génétiques, des opérations comme l'évaluation, le croisement et la mutation peuvent se faire en parallèle. Il serait donc intéressant de proposer une version parallèle de notre algorithme afin d'augmenter sa vitesse.

Bibliographie

[1] V. Derrien. "Heuristiques pour la résolution du problème d'alignement multiple". Thèse de doctorat, Université d'Angers, 2008.

[2] D. Robert et B. Vian. "Éléments de biologie cellulaire 3^e édition". Doin, 2004.

[3] Jack M. Harry. "Génétique moléculaire et évolutive". Maloine, 2001.

[4] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, et P. Walter. "Molecular Biology of the Cell". Garland Science, 4ème édition 2002.

[5] S. Batzoglou . "Sequence Alignment I". CS262 Lecture Notes, 2004.

[6] A. Layeb. "Approche quantique évolutionnaire pour l'alignement multiple de séquences en bioinformatique". Mémoire de magistère, Université Mentouri de Constantine, 2005.

[7] N.M. Luscombe, D. Greenbaum, et M. Gerstein. "What is bioinformatics ? An introduction and overview ". Yearbook of Medical Informatics USA, 2001.

[8] E.P.C. Rocha. "Analyse exploratoire des génomes bactériens" . Thèse de doctorat, Université de Versailles, 2000.

[9] S.B. Needleman et C.D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology, 1970.

[10] C. Notredame. "Recent progresses in multiple sequence alignment : a survey". Pharmacogenomics, 2002.

[11] I.M.Wallace, G. Blackshields, et D.G. Higgins. "Multiple sequence alignments". Current Opinion in Structural Biology, 2005.

[12] W.M. Fitch et E. Margoliash. "Construction of phylogenetic trees". Science,1967.

[13] A. Bairoch. "Prosite : a dictionnary of sites and patterns in proteins". Nucleic Acids Research,1991.

[14] P.Y. Chou et G.D. Fasman. "Prediction of protein conformation". Biochemistery, 1974.

[15] N. Benlahrache. "Optimisation Multi-Objectif Pour l'Alignement Multiple de Séquences". Mémoire de magistère, Université Mentouri de Constantine, 2007.

[16] A.D. Baxevanis. "The molecular biology database collection : an online compilation of relevant database resources". Nucleic Acid Research, 2000.

[17] M.Crochemore, C.Hancart et T.Lecroq . "Algorithmique du texte ". Vuibert, 2001.

[18] C.Charras et T.Lecroq. "Handbook of Exact String Matching Algorithms". King's College Publications, 2004.

[19] M.Crochemore et W.Rytter. "JEWELS OF STRINGOLOGY". World Scientific Publishing, 2002.

[20] D.Beauquier, J.Berstel et Ph.Chrétienne. "Éléments d'algorithmique". Masson, 2005.

[21] D.E. Knuth, J.H. Morris Jr et V.R. Pratt. "Fast pattern matching in strings". SIAM J. Comput., Vol. 6 No. 2, 1977.

[22] R.S. Boyer et J.S. Moore. "A fast string searching algorithm". Communications of the ACM, Vol. 20 No. 10, 1977.

[23] K. Chao et L. Zhang. "Sequence Comparison Theory and Methods". Computational Biology Series, Springer, 2009.

[24] J. Xiong. "Essential Bioinformatics". Cambridge University Press, 2006.

[25] R.W. Hamming. "Error detecting and error correcting codes". Bell System Technical Journal, 1950.

[26] P. Clote et R. Backofen. "Computational Molecular Biology - An Introduction". Mathematical and Computational Biology, Wiley, 2000.

[27] V.I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady, 1966.

[28] M.O. Dayhoff, R.M. Schwartz, et B.C. Orcutt. "A model for evolutionary change in proteins". Atlas of Protein Sequence and Structure, Vol. 5, 1978.

[29] S. Henikoff et J.G. Henikoff. "Amino acid substitution matrices from protein blocks". Proceedings of the National Academy of Science, 1992.

[30] J.G. Henikoff et S. Henikoff. "Blocks database and its applications". Methods in Enzymology, 1996.

[31] S.F. Altschul. "Gap costs for multiple sequence alignment". Journal of Theoretical Biology, 1989.

[32] T. Cormen, C. Leiserson, R. Rivest, et C. Stein. "INTRODUCTION À L'ALGORITHMIQUE Cours et exercices 2éme édition". Dunod, Paris, 2004.

[33] T.M. Smith et M.S. Waterman. "Identification of common molecular subsequences". Journal of Molecular Biology, 1981.

[34] B. Haubold et T. Wiehe. "Introduction to Computationl Biology An Evolutionary Approach". Birkhäuser Verlag, 2006

[35] W.R. Pearson et D.J. Lipman. "Improved tools for biological sequence comparison". Proceedings of the National Academy of Sciences, 1988.

[36] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, et D.J. Lipman. "Basic local alignment search tool". Journal of Molecular Biology, 1990.

[37] P. A. Pevzner. "Bio-informatique moléculaire Une approche algorithmique". Springer, 2000.

[38] S.F. Altschul, R.J. Carroll, et D.J. Lipman. "Weights for data related by a tree". Journal of Molecular Biology, 1989.

[39] N. Saitou et M. Nei. "The neighbor-joining method : a new method for reconstructing phylogenetic trees". Molecular Biology and Evolution, 1987.

[40] C. Notredame, L. Holme, et D.G. Higgins. "Coffee : A new objective function for multiple sequence alignmnent". Bioinformatics, Vol. 14 No. 5, 1998.

[41] L. Wang et T. Jiang. "On the complexity of multiple sequence alignment". Journal of Computational Biology, 1994.

[42] H. Carillo et D.J. Lipman. "The multiple sequence alignment problem in biology". SIAM J. APPL. MATH, Vol. 48 No. 5, 1988.

[43] J. Stoye, V. Moulton, et A.W. Dress. "DCA : an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment". Comput. Apll. Biosci. , Vol. 13 No. 6, 1997.

[44] P. Hogeweg et B. Hesper. "The alignment of sets of sequences and the construction of phylogenetic trees. an integrated method". J. Mol. Evol. , 1984.

[45] D.F. Feng et R.F Doolittle. "Progressive alignment and phylogenetic tree construction of protein sequences". Methods in Enzymology, Vol. 183, 1990.

[46] R.R. Sokal et C.D. Michener. "A statistical method for evaluating systemaic relationships". University of Kansas Science Bulletin, Vol. 38, 1958.

[47] J.D. Thompson, D.G. Higgins, et T.J. Gibson. "Clustal w : improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice". Nucleic Acids Research, Vol. 22 No.22, 1994.

[48] M.A. Larkin, G. Blackshields, N.P. Brown, R. Chenna, P.A. McGettigan, H. McWilliam, F. Valentin, I.M. Wallace, A. Wilm, R. Lopez, J.D. Thompson, T.J. Gibson et D.G. Higgins. "Clustal W and Clustal X version 2.0". BIOINFORMATICS APPLICATIONS NOTE, Vol. 23 No 21, 2007.

[49] C. Notredame, D.G. Higgins, et J. Heringa. "T-coffee : A Novel Method for Fast and Accurate Multiple Sequence Alignment". Journal of Molecular Biology, Vol. 302, 2000.

[50] I.M.Wallace, O. O'Sullivan, D.G. Higgins, et C. Notredame. "M-Coffee : combining multiple sequence alignment methods with T-Coffee". Nucleic Acids Research. Vol. 34 No. 6, 2006.

[51] K. Katoh, K. Misawa, K. Kuma, et T. Miyata. "MAFFT : a novel method for rapid multiple sequence alignment based on fast Fourier transform". Nucleic Acids Research, Vol. 30 No. 14, 2002.

[52] R.C. Edgar. "MUSCLE : multiple sequence alignment with high accuracy and high throughput". Nucleic Acids Research, Vol. 32 No.5, 2004.

[53] R.C. Edgar. "Local homology recognition and distance measures in linear time using compressed amino acid alphabets". Nucleic Acids Research, Vol. 32 No. 1, 2004.

[54] M. Kimura. "The Neutral Theory of Molecular Evolution". Cambridge University Press, 1983.

[55] C.B. Do, M.S.P. Mahabhashyam, M. Brudno, et S. Batzoglou. "ProbCons : Probabilistic consistency-based multiple sequence alignment". Genome Research, Vol. 15, 2005.

[56] C. Notredame et D.G. Higgins. "SAGA : Sequence alignment by genetic algorithm". Nucleic Acid Research, Vol. 24 No. 8, 1996.

[57] S.R. Eddy. "Multiple alignment using hidden markov models". In CA AAAI Press, Menlo Park, editor, Third International Conference on Intelligent Systems for Molecular Biology (ISBM), 1995.

[58] T. Riaz et W. Yi, K.B. Li. "A tabu search algorithm for post-processing multiple sequence alignment". Journal of Bioinformatics and Computational Biology, 2005.

[59] X. Xu et X. Lei. "Multiple Sequence Alignment Based on ABC_SA". Proceedings of Artificial intelligence and computational intelligence : Part II , 2010.

[60] J.D. Thompson, F. Plewniak, et O. Poch. "BAliBASE : A benchmark alignments database for the evaluation of multiple sequence alignment programs". Bioinformatics, Vol. 15 No. 1, 1999.

[61] A. Bahr, J.D. Thompson, J.-C. Thirrey et O. Poch. "BAliBASE (Benchmark Alignment dataBASE) : enhancements for repeats, transmembrane sequences and circular permutations". Nucleic Acids Research, Vol. 29 No. 1, 2001.

[62] J.D. Thompson, P. Koehl, R. Ripp, et O. Poch. "BAliBASE 3.0 : Latest developments of the multiple sequence alignment benchmark". PROTEINS : Structure, Function, and Bioinformatics, Vol. 61, 2005.

[63] M. R. Aniba, O. Poch et J. D. Thompson. " Issues in bioinformatics benchmarking : the case study of multiple sequence alignment". Nucleic Acids Research, Vol. 1 No. 11, 2010.

[64] K. Mizuguchi, C.M. Deane, T.L. Blundell, et J.P. Overington, "Homstrad : A database of protein structure alignments for homologous families", Protein Science, Vol. 7, 1998.

[65] G.P.S. Raghava, S.M.J Searle, P.C. Audley, J.D. Barber, et G.J. Barton. "Oxbench : A benchmark for evaluation of protein multiple sequence alignment accuracy". BMC Bioinformatics, Vol. 4 No. 47, 2003.

[66] I. Van Walle, I. Lasters, et L. Wyns. "Sabmark - a benchmark for sequence alignment that cover the entire known fold space". Bioinformatics, Vol. 21 No. 7, 2005.

[67] A. R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, et B. Morgenstern. "DIALIGN-T : an improved algorithm for segment-based multiple sequence alignment". BMC Bioinformatics, Vol. 6 No. 66, 2005.