Heuristiques pour la résolution du problème d'alignement multiple

Vincent Derrien

▶ **To cite this version:**

Vincent Derrien. Heuristiques pour la résolution du problème d'alignement multiple. Informatique [cs]. Université d'Angers, 2008. Français. NNT : . tel-00352784

HAL Id: tel-00352784
https://theses.hal.science/tel-00352784

Submitted on 13 Jan 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université angers

Année 2008
N° d'ordre 885

Heuristiques pour la résolution du problème d'alignement multiple

Thèse de doctorat

Spécialité : Informatique

École Doctorale d'Angers

Présentée et soutenue publiquement

Le 7 mars 2008

À Angers

Par **Vincent** Derrien

Devant le jury ci-dessous :

Président :	Bernard Levrat	Professeur à l'Université d'Angers
Rapporteurs :	Rumen Andonov,	Professeur à l'Université de Rennes 1
	Clarisse Dhaenens,	Professeur à l'Université de Lille 1
Examinateur :	Jacques Nicolas	Professeur à l'Université de Rennes 1
Directeur de thèse :	Jin-Kao Hao,	Professeur à l'Université d'Angers
Co-directeur de thèse :	Jean-Michel Richer,	Maître de Conférences à l'Université d'Angers

Remerciements

Je tiens tout d'abord à remercier vivement Jin-Kao Hao et Jean-Michel Richer pour le suivi régulier qu'ils ont effectué tout au long de ces années ainsi que pour les conseils qu'ils ont su me donner. Leur expérience et leur soutien m'ont permis de mener à bien ce travail.

Je souhaite également remercier vivement Clarisse Dhaenens qui a accepté d'être rapporteur de cette thèse. Sa lecture minutieuse du manuscrit m'a permis d'apporter bon nombre de précisions facilitant sa lecture et sa compréhension.

Avec sa vision plus biologique du problème, Rumen Andonov a su me poser des questions se détachant du cadre purement informatique. Je tiens à le remercier autant pour le rapport qu'il a effectué que pour les modifications de l'algorithme et de la thèse qu'il a proposées.

Je remercie également Bernard Levrat qui a non seulement accepté de participer à mon jury, mais également de le présider. Son travail sur d'autres problèmes de la bioinformatique lui ont permis de voir ce travail avec un regard extérieur mais néanmoins avisé.

Jacques Nicolas a également accepté de faire partie de mon jury. À ce titre je souhaite également le remercier, même si je regrette qu'il n'ait pu assister à la soutenance.

Je tiens à remercier amicalement David Genest qui m'a aidé à résoudre plusieurs problèmes de programmation, ainsi que quelques soucis informatiques.

Mes remerciements vont également vers tous les membres du laboratoire et du département informatique, et plus particulièrement à Stéphane Loiseau qui m'a permis d'obtenir un financement pour terminer ma thèse. Je souhaite également remercier tous les thésards présents au cours de mes années de thèse, ainsi que les techniciens et les secrétaires.

Je remercie Priscilla Bouas de l'Ecole Doctorale pour sa disponibilité et son écoute lors des petites tracasseries administratives.

Enfin, et non des moindres, je remercie tous mes proches pour leur soutien et leurs encouragements. Je pense en particulier à mes parents, à mes grands-parents et à Yuko ; mais également à ma sœur Jehanne qui a corrigé une partie importante des fautes de ce manuscrit.

À ma grand-mère,

Table des matières

Introduction Générale **1**

Partie I L'alignement de séquences

1 La bioinformatique **7**
1.1 Introduction . . . 7
1.2 Définitions - notions de biologie . . . 8
1.2.1 ADN, ARN et protéines . . . 8
1.3 Représentation informatique . . . 10
1.3.1 Définitions . . . 10
1.3.2 Représentation informatique, format de séquence . . . 11
1.4 Problèmes issus de la bioinformatique . . . 12
1.4.1 Alignement de séquences . . . 12
1.4.2 Phylogénie . . . 12
1.4.3 Recherche de motifs . . . 12
1.4.4 Prédiction de structures . . . 13
1.5 Rappels sur la théorie de la complexité . . . 13
1.5.1 Définition d'un problème . . . 13
1.5.2 Machines déterministe et non-déterministe . . . 14
1.5.3 Les classes de complexité . . . 14
1.5.4 Réduction et complétude . . . 15
1.6 Pourquoi la bioinformatique ? . . . 16
1.7 Conclusion . . . 16

2 Alignement par paires **19**
2.1 Introduction . . . 19
2.2 Généralités . . . 19
2.2.1 Définitions . . . 20
2.2.2 Utilisations de l'alignement par paire . . . 21
2.3 Distance et similarité entre deux séquences . . . 22

2.3.1 Similarité et homologie . . . 22
2.3.2 La distance de Hamming . . . 22
2.3.3 Les opérations d'édition . . . 23
2.3.4 Fonction de score . . . 25
2.3.5 Les matrices de substitution . . . 25
2.3.6 Evaluation des brèches . . . 29
2.3.7 Les fonctions d'évaluation pour alignements par paires . . . 31
2.4 Rappels de programmation dynamique . . . 32
2.4.1 Position du problème : étude d'un exemple simple . . . 32
2.4.2 Principe de la programmation dynamique . . . 34
2.5 Alignement global . . . 34
2.5.1 Généralités . . . 34
2.5.2 Algorithme de Needleman-Wunsch . . . 35
2.6 Alignement local . . . 44
2.6.1 Définition . . . 44
2.6.2 Algorithme de *Smith-Waterman* . . . 45
2.6.3 Résultats et complexité . . . 46
2.6.4 *Blast* : Basic Local Alignment Search Tool . . . 47
2.7 Conclusion . . . 48

3 Alignement multiple de séquences 49
3.1 Généralités . . . 49
3.1.1 Définitions . . . 49
3.1.2 Evaluation d'un alignement multiple . . . 50
3.2 Les fonctions d'évaluation . . . 50
3.2.1 La somme des paires . . . 50
3.2.2 La somme des paires pondérées . . . 51
3.2.3 La fonction *Coffee* . . . 52
3.2.4 Les fonctions de *T-Coffee*, *M-Coffee* et *3D-Coffee* . . . 53
3.2.5 Autres fonctions . . . 54
3.3 Le problème d'alignement multiple de séquences . . . 54
3.3.1 Définition . . . 54
3.3.2 Les utilisations en bioinformatique . . . 55
3.3.3 Classification des algorithmes . . . 55
3.4 Les algorithmes exacts . . . 55
3.4.1 Algorithme basé sur la programmation dynamique . . . 56
3.4.2 Algorithme *MSA* . . . 59
3.4.3 Algorithme *DCA* . . . 60
3.5 Les algorithmes progressifs . . . 61
3.5.1 Présentation . . . 61
3.5.2 Profil et *Guide-Tree* . . . 62
3.5.3 *Clustal W* . . . 66
3.5.4 *MultAlin* . . . 66
3.5.5 *Muscle* . . . 67

3.5.6 *MAFFT* . . . 68
3.5.7 *ProbCons* . . . 69
3.5.8 *Dialign* . . . 69
3.6 Les algorithmes itératifs . . . 69
3.6.1 *SAGA* . . . 70
3.6.2 Autres algorithmes . . . 71
3.7 Récapitulatif des algorithmes . . . 72
3.8 Présentation des jeux d'essai . . . 72
3.8.1 Généralités . . . 72
3.8.2 *Balibase* . . . 73
3.8.3 Les autres bases . . . 76
3.9 Conclusion . . . 76

Partie II Les algorithmes Plasma

4 *Plasma* I : un algorithme d'alignement multiple par insertion de blocs 81
4.1 Présentation générale . . . 81
4.2 La recherche locale . . . 83
4.2.1 Configuration et espace de recherche . . . 83
4.2.2 La fonction d'évaluation . . . 83
4.2.3 Description d'un voisinage . . . 83
4.2.4 La méthode de descente . . . 86
4.3 Complexité de l'algorithme . . . 86
4.4 Résultats expérimentaux . . . 87
4.4.1 Les paramètres du programme . . . 87
4.4.2 Les jeux d'essai . . . 87
4.4.3 Résultats . . . 88
4.5 Conclusion . . . 89

5 *Plasma II* : un algorithme d'alignement multiple de séquences sans profil 91
5.1 Introduction . . . 91
5.2 Principe de l'algorithme . . . 91
5.3 Extension de la programmation dynamique dans *Plasma II* . . . 92
5.4 Algorithme général . . . 95
5.5 Détails de l'implémentation . . . 96
5.5.1 Présentation générale de l'algorithme . . . 96
5.5.2 Principe d'alignement de deux blocs . . . 96
5.5.3 La pondération . . . 101
5.6 Complexité de l'algorithme : . . . 102
5.6.1 Complexité pour les calculs de T_{norm} et T_{gap} . . . 102

5.6.2 Complexité pour le calcul des matrices . . . 102
5.6.3 Complexité globale de l'algorithme . . . 103
5.7 Alignement d'alignements et NP-Complétude . . . 104
5.8 Résultats expérimentaux . . . 104
5.8.1 Présentation des jeux d'essai . . . 105
5.8.2 Conditions expérimentales . . . 105
5.8.3 Résultats . . . 106
5.8.4 Commentaires sur les résultats . . . 107
5.9 Conclusion . . . 108

6 Variantes basées sur l'algorithme de *Plasma II* 111
6.1 Introduction . . . 111
6.2 Utilisation de coûts différents pour les brèches de début et fin d'alignement 112
6.2.1 Introduction . . . 112
6.2.2 Présentation des deux cas . . . 112
6.2.3 Implémentation dans *Plasma II* . . . 113
6.2.4 Résultats . . . 114
6.2.5 Conclusion . . . 116
6.3 Implémentation de la fonction d'évaluation de *Coffee* dans *Plasma II* . . . 116
6.3.1 Rappels sur la fonction d'évaluation de *Coffee* . . . 116
6.3.2 Implémentation dans *Plasma II* . . . 118
6.3.3 Résultats . . . 119
6.3.4 Conclusion . . . 120
6.4 Alignement après construction d'un nouveau *guide-tree* . . . 121
6.4.1 Principe des alignements successifs . . . 121
6.4.2 Implémentation dans *Plasma II* . . . 121
6.4.3 Tests sur les jeux de séquences de *Balibase* . . . 122
6.4.4 Conclusion . . . 122
6.5 Optimisation du résultat par correction des erreurs d'alignements . . . 122
6.5.1 Introduction . . . 122
6.5.2 Présentation des erreurs à corriger . . . 123
6.5.3 Implémentations réalisées dans *Plasma II* . . . 125
6.5.4 Implémentation . . . 125
6.5.5 Tests et résultats . . . 126
6.5.6 Conclusion . . . 126
6.6 Récapitulatif des résultats . . . 127
6.6.1 Introduction . . . 127
6.6.2 Les résultats . . . 127
6.6.3 Conclusion . . . 127
6.7 Conclusion . . . 128

Conclusion Générale **131**

Références bibliographiques **135**

Résumé / Abstract **144**

Introduction Générale

Contexte de travail

Bien que la bioinformatique n'ait utilisé l'informatique à ses débuts que comme un outil pratique de classification de l'information, le volume des données produites par les différents programmes de séquençage des génomes fournit aux biologistes une moisson d'informations sans cesse croissante dont il nous faut extraire la substantifique moelle afin de comprendre les mécanismes de fonctionnement du vivant.

Nous assistons aujourd'hui à une course effrénée dans l'annotation des génomes, l'analyse fonctionnelle des gènes et des protéines ainsi que l'étude à grande échelle de l'interaction des protéines (puces à ADN). Les entreprises privées qui travaillent dans le domaine de la génétique et de la bioinformatique ont d'ailleurs bien compris leur intérêt à s'approprier des fragments de gènes en brevetant leurs découvertes vu le vide juridique actuel.

Dans cette course à la découverte il est nécessaire de disposer d'outils performants et de qualité. Les algorithmes élaborés il y a trente ans, bien que toujours efficaces, nécessitent à présent d'être améliorés tant du point de vue de la qualité que du point de vue de l'efficacité étant donné la masse d'informations à traiter.

Le problème d'alignement de deux séquences d'ADN ou d'Acides Aminés est une opération fondamentale en bioinformatique qui a pour but d'identifier des zones conservées entre deux séquences. On postule que des séquences similaires risquent fort de posséder des propriétés physico-chimiques ou structurelles identiques. Si une nouvelle séquence est similaire à une séquence préalablement étudiée, on sera en mesure d'attribuer une fonction à la nouvelle séquence découverte, puis on vérifiera de manière expérimentale les hypothèses de départ.

Aligner deux séquences consiste à mettre en regard les caractères (nucléotides ou acides aminés) qui se ressemblent. Des caractères ou suites de caractères consécutifs qui semblent similaires dans les deux séquences ne sont pas toujours en regard, et il est donc parfois nécessaire de décaler une partie d'une séquence par rapport à une autre. Cette opération est qualifiée d'insertion de brèche (ou gap en anglais), puisqu'elle permet de visualiser la séparation de la séquence en fragments.

L'insertion de brèches est l'opération de base de l'alignement de séquences. L'objectif est d'en insérer suffisamment pour maximiser les appariements entre les deux séquences, sans toutefois aboutir à un morcellement. Le problème d'alignement de séquences peut donc être vu comme un problème d'optimisation où il faut maximiser le nombre d'appariements tout en minimisant le nombre de brèches introduites. Ce problème peut être résolu de manière exacte à l'aide de la programmation dynamique.

Un autre problème lié à l'alignement de séquences, mais bien plus complexe est le problème d'alignement multiple qui consiste à aligner plus de deux séquences en même temps. L'alignement multiple permet notamment d'étudier un groupe de protéines apparentées afin d'exhiber des motifs communs sensés jouer un rôle dans la fonction ou la

structure des protéines d'une même famille. Ce problème est NP-complet, en d'autres termes il fait partie d'une classe de problèmes difficiles à résoudre.

On peut étendre l'algorithme exact d'alignement de deux séquences pour l'appliquer à l'alignement multiple. On est alors garanti d'obtenir une solution optimale du point de vue de la fonction objectif. Cependant la complexité de l'algorithme qui en résulte est telle qu'on ne peut l'appliquer qu'à un petit nombre de séquences de faible longueur.

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

- Les algorithmes exacts sont basés sur une généralisation de l'algorithme utilisé pour deux séquences. Il est possible d'aligner une vingtaine de séquences en réduisant et en décomposant le problème.
- 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. Cette catégorie regroupe les algorithmes donnant les meilleurs résultats.
- 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.

Motivations et contributions

De nombreux algorithmes ont été développés mais aucun d'entre eux ne permet d'obtenir la solution optimale dans tous les cas. L'existence de jeux d'essais, comme *Balibase*, permet une évaluation plus facile des nouveaux algorithmes. La solution optimale étant proposée avec chaque jeu d'essais, le résultat obtenu par un algorithme peut être évalué par une comparaison directe.

Dans cette optique nous avons développé deux algorithmes différents pour l'alignement multiple de séquences. Il s'agit de deux algorithmes de type progressif, dont l'objectif est de pouvoir aligner ensemble deux alignements.

Le premier, appelé *Plasma* (Progressive LocAl Search for Multiple Alignment), propose pour cela une méthode d'alignement par insertion de colonnes complètes de brèches. L'algorithme cherche à placer au mieux ces brèches dans un des deux groupes de séquences alignées ou dans les deux simultanément. Le processus se termine lorsqu'il n'est plus possible d'améliorer la qualité du résultat.

Le second algorithme, baptisé *Plasma II*, est une méthode qui permet de résoudre un problème proposé il y a quelques années [Kececioglu and Zhang, 1998] : l'alignement d'alignements. Le même auteur propose dans [Kececioglu and Starrett, 2004] une méthode de résolution difficile à cerner et qui semble peu efficace.

Notre algorithme permet d'apporter une réponse à ce problème avec une complexité moindre en utilisant une méthode basée sur la programmation dynamique. Les tests effectués sur les jeux d'essais de *Balibase* montrent que notre algorithme donne de bons résultats, sans toutefois égaler les meilleurs algorithmes actuels. Nous avons donc implémenté des modifications sur notre algorithme, afin de pallier certaines erreurs d'alignement.

Les tests effectués sur les jeux d'essais de *Balibase* montrent que notre algorithme donne de bons résultats, sans toutefois égaler les meilleurs algorithmes actuels.

[Kececioglu and Starrett, 2004; Wheeler and Kececioglu, 2007] proposent une méthode de résolution basée sur l'évaluation pessimiste / optimiste des brèches dont la complexité théorique pour aligner 2 alignements de k séquences de longueur n est dans le pire des cas en $O(5^k.n^2)$, mais en pratique en $O(k^2.n^2)$. Notons que Kececioglu indique que ce problème est NP-Complet (même s'il trouve cela surprenant), or le modèle que nous avons mis en place, basé sur la programmation dynamique, est en $O(k^2.n^2)$. Kececioglu a également développé un logiciel *alignalign* qui est censé calculer de manière exacte le résultat de l'alignement de deux alignements par rapport à la somme des paires en utilisant un fonction de gap affine. Cependant des expérimentations que nous avons pu mener montre que par rapport à Plasma II, alignalign ne trouve que très rarement la solution optimale (5 %).

Organisation de la thèse

Le manuscrit se décompose en deux parties. La première présente un état de l'art pour le problème d'alignement de séquences. Le premier chapitre est axé sur la bioinformatique, afin de repositionner le problème de l'alignement de séquences, mais également afin de définir les notions biologiques utilisées par la suite. Les deux chapitres suivants présentent en détail le problème de l'alignement de séquences. Le premier est consacré au cas particulier de l'alignement de deux séquences. Nous y exposons le problème ainsi que des algorithmes exacts pour le résoudre. Nous terminons cette partie par la présentation du problème d'alignement multiple de séquences proprement dit. Ce problème étant NP-Complet, nous exposons quelques uns des algorithmes représentatifs permettant d'y apporter des solutions.

La seconde partie est consacrée aux apports personnels. Nous présentons tout d'abord *Plasma* dans un premier chapitre. Les deux chapitres suivants sont consacrés à *Plasma II*, tout d'abord pour présenter l'algorithme proprement dit ainsi que les résultats que nous avons obtenus. Dans un deuxième temps nous présentons quelques modifications qui comme nous le verrons nous permettent d'obtenir de bons résultats. L'algorithme *Plasma II* ainsi que les différentes modifications ont été testés sur les jeux d'essais de *Balibase*, en utilisant les deux critères SPS et CS qui permettent d'évaluer la qualité d'un alignement par rapport à un alignement de référence.

Première partie

Chapitre 1

La bioinformatique

1.1 Introduction

C'est en 1866, après plusieurs années de recherches, que Mendel énonce les 3 lois de l'hérédité qui sont à la base de la génétique moderne. Au début du XX$^{\text{ème}}$ siècle sont découverts successivement les chromosomes, par observation microscopique des noyaux de cellules, et les gènes. Il faudra attendre la fin de la seconde Guerre Mondiale pour découvrir tout d'abord la structure des gènes sous forme d'ADN, puis quelques années après la structure même de cet ADN [Watson and Crick, 1953]. A partir de là, la cadence des découvertes va en accélérant. Les étapes principales sont le début de la manipulation de l'ADN en 1970, donnant ainsi naissance à une nouvelle discipline : le génie génétique. Depuis ces vingt dernières années, les séquençages de génomes se sont multipliés. Pour arriver finalement en 2003 jusqu'au séquençage complet du génome humain. Plusieurs projets, tels que *HUGO* (http ://www.hugo-international.org), voient le jour pour parvenir à ce séquençage.

L'analyse de séquences est un concept apparu vers la fin des années 60, soit un siècle après les lois de Mendel. En effet, les découvertes réalisées en biologie à partir de cette date vont apporter de plus en plus d'informations sur la structure des séquences. En quelques années ont ainsi été créées les bases de ce qui deviendra par la suite la bioinformatique. Citons par exemple quelques articles fondateurs : [Fitch and Margoliash, 1967] pour la reconstruction de phylogénie, [Needleman and Wunsch, 1970] pour l'alignement de séquences et [Chou and Fasman, 1974] pour la prédiction de structures secondaires.

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 on ne considère plus la bioinformatique comme uniquement un traitement informatique de problèmes biologiques. La bioinformatique est en effet devenue une science à part entière faisant appel aux compétences de plusieurs disciplines. Différents points de vue permettent de considérer également l'analyse de séquences comme des problèmes mathématiques, chimiques ou bien encore physiques.

En dehors de l'aspect algorithmique proprement dit, bon nombre de disciplines de l'in-

formatique sont impliquées dans le traitement des données biologiques. Citons par exemple les bases de données, l'extraction de connaissances, la programmation par contraintes ou bien encore le traitement du langage naturel.

Un minimum de connaissances en biologie est nécessaire pour la lecture de ce manuscrit. Dans ce chapitre, nous allons tout d'abord rappeler les différentes notions qu'il est indispensable de connaître. Ces notions concernent plus particulièrement les séquences, ainsi que les constituants qui les composent : ADN, ARN et protéines. Nous exposerons ensuite quelques uns des principaux domaines de recherche de la bioinformatique, liés à l'analyse des séquences.

1.2 Définitions - notions de biologie

La médiatisation faite autour du séquençage des génomes, et plus particulièrement du séquençage du génome humain rend familiers certains des termes utilisés. Ainsi l'utilisation d'ADN, protéine ou bien encore séquence, est devenue familière pour tout un chacun. Cependant les notions auxquelles ces termes se rapportent sont souvent un peu floues. Nous allons donc faire ici quelques rappels en donnant des définitions précises.

1.2.1 ADN, ARN et protéines

L'ADN : acide désoxyribonucléique

L'ADN est un acide de la famille des acides nucléiques, composé d'une succession de nucléotides. On peut définir [Harry, 2001] l'ADN d'un organisme comme l'ensemble des informations génétiques nécessaires à l'édification, au fonctionnement et à la reproduction de chaque organisme.

Définition 1 (Nucléotides) *Les nucléotides sont constitués à partir d'une base azotée (Adénine, Thymine, Guanine, Cystosine) d'un sucre et d'un groupement phosphate. Par convention on utilise la première lettre de chacune des bases pour appeler les nucléotides.*

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. 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 :

- $A \iff T$
- $C \iff G$

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

Définition 2 (Séquence d'ADN) *On appelle séquence d'ADN la lecture séquentielle de l'ensemble des bases constituant un brin d'ADN.*

Définition 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.*

Définition 4 (Longueur) *On appelle longueur d'une séquence d'ADN le nombre de bases qui composent cette séquence.*

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.

Définition 5 (Transcription) *La transcription est le mécanisme qui permet de dupliquer un brin d'ADN sous forme d'ARN.*

L'ARN est subdivisé en trois catégories, chacune ayant un rôle différent.
- L'*ARN de transfert* (ARN_t) est utilisé dans la phase de *traduction* de l'ARN en protéines.
- L'*ARN ribosomique* (ARN_r) forme une trame sur les ribosomes, afin de permettre aux protéines synthétisées par les ribosomes de se fixer.
- L'*ARN messager* (ARN_m) est utilisé chez les eucaryotes pour véhiculer l'information génétique du noyau vers le cytoplasme (substance de la cellule qui entoure le noyau).

Les protéines

En prenant les bases de l'ARN par groupes de trois, on obtient les *codons*. Il existe donc $3^4 = 64$ possibilités pour former les codons. Trois des codons sont utilisés pour stopper la synthèse des protéines et les 61 restants servent à coder les acides aminés.

A plusieurs codons peut correspondre le même acide aminé. Au total, il existe 20 acides aminés, représentés par un alphabet de 20 lettres dont la liste est donnée dans le tableau 1.1.

Définition 6 (Protéine) *Une protéine est une séquence composée d'acides aminés.*

Définition 7 (Traduction) *La traduction est le mécanisme qui permet de synthétiser sous forme de protéine l'ARN_m obtenu par transcription.*

$$ADN \longrightarrow ARN_m \longrightarrow Protéines$$

Définition 8 (Gène) *Chaque protéine résulte d'une séquence d'ADN appelée un gène.*

Définition 9 (Structure secondaire) *La structure secondaire décrit le repliement local de la chaîne principale d'une protéine.*

La structure secondaire des protéines est différente de la structure secondaire en double hélice de l'ADN.

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 10 (Structure tertiaire) *La structure tertiaire (3D), ou structure tridimensionnelle, d'une protéine correspond au repliement de la chaîne polypeptidique dans l'espace.*

L'étude de cette structure est aussi importante que celle des structures primaire et secondaire. En effet, la structure tertiaire d'une protéine joue un grand rôle dans sa fonction. Si celle-ci est modifiée, la protéine est alors dénaturée et elle perd sa fonction.

1.3 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.

1.3.1 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 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 12 (Séquence) *On appelle séquence S sur un alphabet Σ une suite ordonnée d'éléments appartenant à Σ, $S = \langle x_1, x_2, \dots x_n \rangle$.*

Définition 13 (Longueur) *On appelle longueur d'une séquence le nombre d'éléments qui la composent. On la note $|S| = n$.*

Définition 14 (Sous-séquence) *Soit S une séquence de longueur n. On appelle 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 $S = \langle x_i, \ldots x_j \rangle$. Nous avons en particulier $S[i..i] = S[i] = \langle x_i \rangle$.*

Définition 15 (Préfixe) *Soit S une séquence de longueur n. On appelle préfixe de S de longueur p toute sous-séquence $S[1..p]$, avec $1 \leq p < n$.*

1.3.2 Représentation informatique, format de séquence

Représenter un alphabet en informatique est 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.

Pourquoi dans ce cas chercher à avoir différents formats de séquence alors que tout semble évident ? Tout d'abord, il est important de se rappeler quels éléments sont manipulés. En effet, lorsque l'on parle de séquence, il s'agit d'une toute petite partie d'un des très nombreux génomes existants. Il est donc normal de pouvoir identifier les séquences de façon unique, mais également de pouvoir si on le souhaite y joindre des informations. Celles-ci peuvent aussi bien concerner la date de la découverte que la nature de la protéine ou sa structure.

De nombreuses séquences sont disponibles librement sur Internet, souvent conservées dans des bases de données avec les informations qui leur sont relatives. Dans [Baxevanis, 2000] sont répertoriées plus de 200 bases de données, classées par catégories de séquences. Certains sites, comme par exemple celui du NCBI (National Center for Biotechnology Information, `http ://www.ncbi.nlm.nih.gov/`), proposent des génomes entiers en libre accès.

Il existe de nombreux formats de séquences : plus d'une trentaine sont répertoriés et utilisés. 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.

Une même séquence peut donc avoir un format de représentation plus ou moins riche et plus ou moins complexe. Ainsi, le format *Fasta* [Pearson and Lipman, 1988] ne contient que peu d'informations, principalement le nom et la séquence elle-même. Ce format est toutefois suffisant pour des problèmes tels que l'alignement de séquences. À l'opposé, le format utilisé par la *Protein Data Bank* [Bernstein *et al.*, 1977] contient, entre autres, la

position de chaque atome dans un repère à trois dimensions, la structure cristallographique ou encore des résultats expérimentaux.

1.4 Problèmes issus de la bioinformatique

Nous présentons ici quelques uns des principaux domaines de la bioinformatique, liés aux définitions que nous venons de voir [Gibas and Jambeck, 2002].

1.4.1 Alignement de séquences

L'alignement de séquences est une problématique importante de la bioinformatique [Needleman and Wunsch, 1970], [Notredame, 2002], [Wallace *et al.*, 2005]. 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 dans les deux chapitres suivants de ce manuscrit.

1.4.2 Phylogénie

Les arbres phylogénétiques [Fitch and Margoliash, 1967] 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. Autrement dit il est préférable d'avoir des séquences *orthologues*.

1.4.3 Recherche de motifs

Un motif (ou Pattern) au sens bioinformatique du terme [Bairoch, 1991], 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 et 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 étapes dans la recherche de motif :

– la découverte 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 des séquences afin de trouver un motif simple.
– la recherche à proprement parler, qui concerne la détection d'un motif donné sur un ensemble de séquences. Ce problème est bien plus simple que le premier.

Les deux problèmes rencontrés dans la recherche de motifs concernent la définition du motif. Il existe une nomenclature disponible sur le site de *Prosite* [Hulo *et al.*, 2006] pour la description des motifs.

1.4.4 Prédiction de structures

Le nombre de structures primaires possibles pour les protéines est exponentiel en fonction de la longueur l. 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 [Chou and Fasman, 1974] sans utiliser une analyse directe de la séquence. 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. Il est nécessaire pour cela de trouver des séquences similaires. Comme nous le verrons dans le chapitre III, des outils ont été développés pour ce genre d'opérations.

1.5 Rappels sur la théorie de la complexité

La complexité des problèmes étudiés en bioinformatique nécessite de faire quelques rappels. Nous présentons ici les définitions nécessaires à l'introduction de la classe des problèmes *NP-complets*. Pour plus de détails, le lecteur peut consulter les ouvrages suivants : [Garey and Johnson, 1979], [Wilf, 1989], [Turing, 1995] et [Cori and Lascar, 2003].

1.5.1 Définition d'un problème

On appellera *problème de décision* tout problème pour lequel il est possible de répondre par oui ou non. En informatique, tout problème peut être transformé en un problème de décision.

La résolution d'un problème peut être réalisé en utilisant une *machine de Turing* [Turing, 1950]. Une machine est une modélisation théorique pouvant s'apparenter à un ordi-

nateur. Elle est en mesure d'effectuer différentes opérations élémentaires en fonction de l'état dans lequel se trouve sa mémoire.

1.5.2 Machines déterministe et non-déterministe

Il est possible de différencier deux catégories de machines de Turing.

Définition 16 (Machine déterminisite) *Une machine est dite déterministe lorsqu'elle vérifie les critères suivants :*
- *elle effectue les opérations élémentaires une par une,*
- *à chaque étape de son calcul, la machine n'a qu'une opération possible à effectuer.*

A partir d'un problème donné, une machine déterministe effectuera donc toujours la même suite d'opérations élémentaires. Elle donnera donc toujours en sortie le même résultat.

Définition 17 (Machine non-déterministe) *Une machine est dite non-déterministe lorsqu'il existe au moins une étape où elle a le choix entre plusieurs opérations à effectuer.*

Lorsque l'on parle de complexité en informatique il est primordial de faire la différence entre la complexité en temps et la complexité en espace. La complexité en temps correspond à une approximation de la durée des calculs nécessaires pour qu'une machine réponde à un problème de décision. La complexité en espace correspond à l'espace mémoire nécessaire pour effectuer les mêmes calculs.

1.5.3 Les classes de complexité

S'il vérifie certains critères, chaque problème peut être rangé dans une ou plusieurs classes. Ces classes permettent donc de mettre en place des catégories de problèmes, en prenant comme critère le temps ou l'espace nécessaire pour qu'une machine réponde à un problème de décision.

Voici les classes de complexité les plus souvent rencontrées.

Définition 18 (P) *La Classe P correspond aux problèmes pour lesquels il existe un algorithme donnant une réponse en temps polynomial sur une machine déterministe.*

Les problèmes de la classe P peuvent donc être résolus avec des algorithmes de complexité polynomiale.

Définition 19 (NP) *La Classe NP correspond aux problèmes pour lesquels il existe un algorithme donnant une réponse en temps polynomial sur une machine non-déterministe.*

Il est évident que les problèmes de P appartiennent à NP en effet, s'il existe un algorithme polynomial sur une machine déterministe, cet algorithme est également polynomial sur une machine non déterministe. En revanche, l'inclusion ou la non-inclusion réciproque n'a pas été démontrée. Il est impossible de conclure sur l'égalité ou non des classes P et NP.

Définition 20 (PSpace) *La Classe PSpace correspond aux problèmes pour lesquels il existe un algorithme donnant une réponse en espace polynomial sur une machine déterministe.*

Définition 21 (NSpace) *La Classe NSpace correspond aux problèmes pour lesquels il existe un algorithme donnant une réponse en espace polynomial sur une machine non-déterministe.*

Les autres classes correspondent principalement à des réponses logarithmiques ou exponentielles en temps ou en espace sur des machines deterministes ou non.

1.5.4 Réduction et complétude

Définition 22 (Réduction) *Soient P_1 et P_2 deux problèmes. On dit qu'un algorithme est une réduction de P_1 vers P_2 s'il permet de transformer une solution de P_1 en une solution de P_2.*

Une des réductions de problèmes les plus utilisées est la réduction polynomiale : c'est-à-dire que le nombre d'opérations nécessaires pour réaliser la réduction peut être exprimée de façon polynomiale.

Définition 23 (Problème C-Complet) *Soit C une classe de complexité. On dit qu'un problème P_1 est C-Complet s'il vérifie les deux critères :*
- *P_1 est dans la classe C,*
- *il existe un problème P_2 de C pouvant être réduit vers P_1.*

Dans le cas de la classe NP, nous avons la définition suivante :

Définition 24 (Problème NP-Complet) *On dit que P_1 est un problème NP-Complet s'il est dans la classe NP et s'il existe un problème P_2 de NP pouvant être réduit de façon polynomiale vers P_1.*

Cette définition permet donc de construire progressivement l'ensemble des problèmes *NP-Complets*. Mais par définition pour ajouter un problème dans cet ensemble, il est nécessaire qu'il ne soit pas vide. Il a donc été nécessaire initialement de trouver un problème dans *NP*.

Définition 25 (Littéral) *Un littéral est une variable booléenne qui peut prendre soit la valeur vrai soit la valeur faux.*

Définition 26 (Clause d'ordre n) *Une clause d'ordre n est une disjonction d'au plus n littéraux. C'est-à-dire qu'elle peut être écrite sous la forme : $l_1 \vee l_2 \vee \dots l_n$, où les l_i représentent des littéraux.*

Définition 27 (Forme Normale Conjonctive) *On appelle Forme Normale Conjonctive d'ordre n (n-CNF) toute disjonction de clauses d'ordre n. Une n-CNF peut donc s'écrire sous la forme $c_1 \wedge c_2 \wedge \dots c_m$ où les c_i représentent des clauses d'ordre n.*

Définition 28 (Formule satisfiable) *Une formule est satisfiable si et seulement si elle est vraie pour au moins une affectation des littéraux.*

Définition 29 (Problème Sat) *On appelle problème Sat d'ordre n un problème qui consiste à déterminer si une formule n-CNF est satisfiable.*

Le *Théorème de Cook* [Cook, 1971] établit que le problème *Sat* est *NP-Complet*. Il devient alors possible par réduction polynomiale à *Sat* de montrer que tout problème est *NP-Complet*.

Pour cette raison, *Sat* est un problème très étudié. Trouver un algorithme polynomial pour résoudre *Sat*, reviendrait à démontrer que $P = NP$.

1.6 Pourquoi la bioinformatique ?

Les problèmes que nous avons présentés (alignement, phylogénie, ...) ont tous une grande importance en biologie moléculaire, ils ont toutefois un autre point commun. En effet chacun de ces problèmes a une forte complexité, ce qui implique qu'il est bien souvent impossible de les résoudre de manière exacte.

Chacun de ces problèmes a été démontré comme étant *NP-Complet*. Il est possible de calculer des solutions pour de petites instances au moyen d'algorithmes exacts, toutefois d'autres méthodes doivent être envisagées pour les instances de plus grande taille. En effet, la forte complexité des algorithmes exacts ne permet bien souvent pas de les utiliser avec les données biologiques de grandes tailles.

Devant l'importance des problèmes, il est nécessaire de pouvoir obtenir des résultats de bonne qualité pour qu'ils puissent être utilisés par les biologistes. L'objectif de la bioinformatique est donc de réunir les compétences disponibles dans chaque discipline scientifique afin de contribuer à l'amélioration des méthodes de résolution. Par exemple pour le problème d'alignement multiple de séquences, des solutions informatiques et mathématiques ont été apportées.

Les solutions apportées pour ces problèmes sont souvent données sous forme d'un algorithme. Bon nombre de programmes sont mis gratuitement à la disposition de la communauté en téléchargement. Mais il est souvent également possible de les utiliser directement en ligne à partir d'un ou plusieurs sites Internet (http ://www.ebi.ac.uk/).

1.7 Conclusion

Dans ce chapitre nous avons fait quelques rappels sur l'histoire de la biologie moléculaire. Et nous pouvons constater l'accélération des découvertes depuis les années 1970. La quantité d'informations disponibles augmente dans les mêmes proportions, impliquant ainsi toujours plus de données à traiter. Or les domaines d'étude classique de la biologie ne permettent pas de les traiter de façon efficace.

Un des objectifs de la bioinformatique est de mettre en œuvre toutes les connaissances disponibles dans les différentes disciplines scientifiques, afin d'aider à comprendre le fonctionnement des organismes vivants du point de vue moléculaire.

La bioinformatique est une discipline récente, qui fait appel aux compétences de la plupart des disciplines scientifiques pour lesquelles il existe déjà des méthodes permettant de résoudre des problèmes analogues. 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.

La raison même de l'existence de la bioinformatique réside dans la difficulté des problèmes auxquels elle est confrontée. Nous en avons présenté quelques uns, qui sont des problèmes très courants de la biologie moléculaire (alignement de séquences, reconstruction de phylogénie, etc.), et pour lesquels il est primordial pour les biologistes de pouvoir obtenir des résultats. Le point commun de ces problèmes est leur complexité. Les méthodes exactes permettant de les résoudre ne sont bien souvent utilisables que pour des petites instances. En pratique, ce sont les méthodes approchées qui sont employées pour apporter des solutions de bonne qualité.

Chapitre 2

Alignement par paires

2.1 Introduction

L'alignement de deux séquences, également appelé alignement par paires, est un cas particulier du problème d'alignement multiple de séquences que nous aborderons au chapitre suivant. L'alignement par paires a été très étudié non seulement parce qu'il s'agit d'une simplification de ce dernier problème plus général, mais également parce qu'il peut faire l'objet d'applications directes.

Comme nous le verrons par la suite, l'alignement de deux séquences est à la base de plusieurs algorithmes d'alignement multiple de séquences. Mais l'alignement de deux séquences permet également de définir facilement une distance et une similarité entre deux séquences. Il offre ainsi un critère pour une recherche dans une base de données composée de nombreuses séquences.

Dans ce chapitre, nous allons présenter le problème d'alignement de deux séquences dans son ensemble. Pour cela nous allons présenter le problème de façon globale, en expliquant le principe et les utilisations qui peuvent en être faites. Même si aligner deux séquences semble être un problème difficile à résoudre, il existe un algorithme exact de complexité polynomiale pour ce problème. Toutefois lorsqu'il est nécessaire de calculer de nombreux alignements, cette complexité peut s'avérer trop importante pour que l'algorithme soit utilisable.

2.2 Généralités

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. Pour faciliter la lecture on matérialise les coupures au sein des séquences par le symbole '-'. Par exemple pour aligner les deux séquences suivantes :

- S_1 : `GACTGAG`
- S_2 : `GCTGGAAG`

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 $\Sigma' = \Sigma \cup \{-\}$:

$$S'_1 : \texttt{GACTG--AG}$$
$$S'_2 : \texttt{G-CTGGAAG}$$

2.2.1 Définitions

Nous allons ici formaliser le concept d'alignement de deux séquences [Giegerich and Wheeler, 1996] présenté ci-dessus. La première définition que nous donnons concerne la représentation des fractures dans les séquences.

Définition 30 (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.*

Le concept de brèche dans une séquence étant établi, cela nous permet maintenant de donner une définition formelle de l'alignement de deux séquences. Nous commençons tout d'abord par définir une fonction permettant de retouver la séquence initiale.

Définition 31 (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 32 (Ote séquence) *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 33 (Alignement de deux séquences) *Soient $S = \langle x_1, \ldots, x_m \rangle$ et $T = \langle y_1, \ldots, y_n \rangle$ 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 $ote(T') = T$,*
- *$\nexists i / S'[i] = T'[i] =$ '-'.*

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. 2.1 – *Nombre d'alignements possibles de 2 séquences.*

Remarque 1 *Compte tenu de la définition donnée ci-dessus, la longueur p de l'alignement des deux séquences S et T vérifie la propriété suivante :*

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

De cette inégalité, on peut déduire facilement que le nombre d'alignements possibles pour les deux séquences S et T est fini.

Proposition 1 (Dénombrement d'alignements) *Soient S et T deux séquences de même longueur n. Le nombre N d'alignements possibles de S et T est donné par :*

$$N = \sum_{k=0}^{n} C_{n+k}^{k}.C_{n}^{k}$$

Une simple application numérique permet de se rendre compte du nombre élevé d'alignements possibles pour des séquences de longueurs assez courantes. Les résultats sont donnés dans le tableau 2.1 pour des valeurs de n comprises entre 50 et 1400. L'objectif de ce tableau est de montrer la complexité du problème pour des longueurs de séquences rencontrées fréquemment. Une longueur de 50 correspond en effet à une petite séquence, et il n'est pas rare d'avoir des séquences de longueur supérieure à 1000.

L'hypothèse de longueurs identiques pour les deux séquences permet de simplifier la formule. Celle-ci n'étant ici donnée qu'à titre indicatif, afin de pouvoir comprendre la difficulté du problème.

Comme nous le verrons par la suite, même si le nombre d'alignements possibles est très important, il existe une méthode efficace permettant de déterminer un alignement optimal suivant un critère donné.

2.2.2 Utilisations de l'alignement par paire

L'alignement par paire de séquences est utilisé pour plusieurs problèmes de la bioinformatique. Comme il s'agit d'un problème assez simple à résoudre et assez peu coûteux en temps, il est fréquemment employé soit comme traitement à part entière, soit comme prétraitement.

- La première utilisation courante de l'alignement par paire est la comparaison de séquences. Le résultat de l'alignement des deux séquences permet de faire une comparaison afin de déterminer si deux séquences sont semblables ou non, et si elles ont des parties communes.
- En deuxième application nous donnons la recherche dans une base de données. Pour étudier une nouvelle séquence, on peut chercher si elle est ou non déjà répertoriée dans les bases de données. Il se peut que la séquence soit tronquée ou soit une partie d'une séquence déjà existante. Il n'est donc pas possible de rechercher une identité parfaite entre la nouvelle séquence et les séquences des bases. Une requête faite en utilisant comme critère l'alignement par paire va permettre de sélectionner des séquences similaires.
- Enfin, l'alignement par paire est souvent utilisé, comme nous le verrons dans le chapitre suivant, pour l'alignement multiple de séquences. Tout d'abord, en réalisant les alignements par paires de tous les couples de séquences il va être possible de déterminer leurs degrés de similitude. Ensuite, certaines méthodes d'alignement multiple sont basées sur le principe d'alignement par paire. En effet, en convertissant un ensemble de séquences en une unique séquence consensus, aligner deux groupes de séquences revient à aligner leurs deux séquences consensus. L'algorithme est alors utilisé de façon itérative afin de créer l'alignement complet de toutes les séquences.

2.3 Distance et similarité entre deux séquences

2.3.1 Similarité et homologie

La similarité entre deux séquences peut être expliquée en prenant comme postulat de départ que toutes les espèces vivantes sont issues d'un même ancêtre. Selon cette théorie, des mutations interviennent au niveau de l'ADN, générant ainsi de nouvelles espèces. Ces mutations se produisent localement et peuvent être de différents types :

- suppression d'un ou plusieurs nucléotides,
- insertion d'un ou plusieurs nucléotides,
- mutation d'un nucléotide en un autre.

Au sens de la théorie de l'évolution, deux séquences peuvent donc être plus ou moins proches, selon le nombre de modifications ayant eu lieu.

Définition 34 (Homologie et similarité) *On dit qu'il y a homologie entre deux séquences lorsque celles-ci possèdent une parenté du point de vue de l'évolution. 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.*

2.3.2 La distance de Hamming

Définition 35 *Soient S et T deux séquences sur un alphabet Σ. On dit que deux lettres s_i et t_i de S et T se correspondent si, et seulement si, $s_i = t_i$. On utilise également le terme anglais match pour indiquer la correspondance de deux lettres.*

Définition 36 (Distance de Hamming) *Soient S et T deux séquences de même longueur n sur un alphabet Σ. La distance de Hamming [Hamming, 1950] 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 :*

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

Cette formule compte le nombre de positions où les lettres ne se correspondent pas. Le cas particulier $n = 1$ permet de définir la *distance de Hamming* entre deux lettres. Elle peut donc également s'écrire :

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

Exemple : Soient S et T les deux séquences suivantes :

```
S : ACACACAT
T : CACACAT
```

La distance de Hamming entre $S[1..7]$ et T est donc $d_H(S[1..7],T) = 7$, alors que la distance entre $S[2..8]$ et T est $d_H(S[2..8],T) = 0$.

Remarque : Cette formule permet bien de définir une distance au sens mathématique. En effet les propriétés de *symétrie* et *séparation* sont évidentes. On peut montrer aisément l'*inégalité triangulaire* en prenant une troisième séquence U de longueur n également. Deux cas sont alors possibles :

- Lorsque $d_H(S[i],T[i]) = 0$, nous avons obligatoirement $d_H(S[i],T[i]) \leq d_H(S[i],U[i]) + d_H(U[i],T[i])$,
- Lorsque $d_H(S[i],T[i]) = 1$, on ne peut pas avoir en même temps $S[i] = U[i]$ et $T[i] = U[i]$. Donc $d_H(S[i],U[i]) + d_H(U[i],T[i]) \geq 1$, et donc $d_H(S[i],T[i]) \leq d_H(S[i],U[i]) + d_H(U[i],T[i])$.

En faisant la somme, on obtient finalement :

$$d_H(S,T) \leq d_H(S,U) + d_H(U,T)$$

2.3.3 Les opérations d'édition

Définition des opérations d'édition

L'alignement de deux séquences consiste à mettre en regard tous les caractères composant ces deux séquences. Les opérations d'éditions [Clote and Backofen, 2000] 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 : (a,a),

S : AGTGAGG	S' : AG--TGAGG
T : AGGTTGCG	T' : AGGTTG-CG

FIG. 2.1 – Exemple d'alignement.

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

L'évolution de la séquence S jusqu'à la séquence T s'obtient en réalisant des deletions à la place des insertions, et réciproquement.

Exemple : Dans l'exemple donné à la figure 2.1, la suite d'opérations d'édition permettant de transformer S' en T' est :

- deux appariements,
- deux insertions,
- deux appariements,
- une deletion,
- une substitution,
- un appariement.

La distance d'édition

Les opérations d'édition permettent de définir une distance entre les deux séquences appelée *distance d'édition* ou *distance de Levenstein* [Levenshtein, 1966]. Pour cela on associe une valeur à chaque opération. L'appariement correspondant à une identité a une valeur de 0, les autres opérations ont une valeur de 1. Pour deux séquences S et T, le nombre de transformations possibles de S en T est fonction de la longueur des deux séquences. Plus les séquences sont longues et plus il existe de façons d'effectuer le passage de S à T. Les coûts associés sont définis comme la somme des coûts de toutes les opérations d'édition utilisées. Le coût est donc variable suivant les opérations utilisées.

Définition 37 (Distance de Levenstein) *Soient S et T deux séquences, et soit $\mathcal{S}$ 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 $\mathcal{S}$.*

Remarque 2 *Les opérations d'édition permettant de transformer T en S sont les mêmes que celles permettant de transformer S en T, en inversant les insertions et les deletions. Ces deux opérations ayant le même coût, la distance de S à T est donc égale à la distance de T à S.*

Comme pour la distance de Hamming, nous avons bien les propriétés de *symétrie*, de *séparation* et l'*inégalité triangulaire*. La distance d'édition ainsi définie est donc bien une distance au sens mathématique.

Remarque 3 *La suite d'opérations permettant d'obtenir la distance d'édition entre deux séquences n'est pas nécessairement unique.*

2.3.4 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. En effet, les opérations d'édition sont réalisées uniquement sur des couples d'éléments, où l'un des deux peut être un $-$ dans le cas d'une *insertion* ou d'une *deletion*.

L'évaluation d'un alignement de séquences que nous avons vue avec la distance d'édition est généralisable. En affectant des coûts à chaque couple d'éléments de $\Sigma \cup \{-\}$ il est possible de définir une évaluation de tout alignement.

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 38 (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 $f : \Sigma'^* \times \Sigma'^* \longrightarrow \mathbb{R}$, et qui a un alignement associe la somme des valeurs de ses opérations d'édition.*

L'utilisation d'une fonction de score nécessite d'avoir les deux éléments que nous allons détailler maintenant : une matrice de substitution permettant d'associer une valeur aux operations d'appariement et de remplacement, ainsi qu'un modèle d'évaluation pour les brèches associant une valeur aux opérations d'insertion et de deletion.

2.3.5 Les matrices de substitution

Les matrices de substitution pour l'ADN

Contrairement aux protéines, les matrices utilisées pour l'ADN sont généralement très simples et n'ont que peu de sens du point de vue biologique. Elles ont souvent uniquement pour rôle d'attribuer une valeur aux différentes configurations possibles. L'évaluation que l'on peut donner à un couple de nucléotides (x, y) est la même que l'évaluation du couple (y, x), toutes les matrices sont donc symétriques. Les matrices de substitution pour l'ADN peuvent être divisées en deux catégories. En effet le problème d'alignement peut être considéré de deux points de vue différents.

Le premier est lié au concept de distance tel que nous l'avons exposé. Le score obtenu est égal à la distance entre les deux séquences. Or pour faire apparaître la similarité de deux séquences il faut minimiser la distance entre ces deux séquences. Il s'agit donc dans ce cas d'un problème de minimisation. Suivant les couples de nucléotides, on associera des valeurs plus ou moins importantes. Comme nous l'avons vu, pour être une distance, il est nécessaire d'associer une valeur nulle à l'identité. Pour les autres couples de nucléotides on donnera une valeur d'autant plus élevée qu'il s'agit d'une configuration à éviter. Il est à remarquer que suivant les valeurs de la matrice de substitution, la fonction associée peut ne plus être une distance. D'une manière générale le score représente une valeur que l'on

	$-$	A	C	G	T
$-$	/	1	1	1	1
A	1	0	1	1	1
C	1	1	0	1	1
G	1	1	1	0	1
T	1	1	1	1	0

TAB. 2.2 – *Matrice de substitution de Hamming*

	$-$	A	C	G	T
$-$	/	1	1	1	1
A	1	6	2	2	2
C	1	2	6	2	2
G	1	2	2	6	2
T	1	2	2	2	6

TAB. 2.3 – *Exemple de matrice de substitution pour un problème de maximisation.*

cherche généralement à maximiser. On parle donc plus souvent de fonction de coût lorsqu'il s'agit d'une fonction que l'on souhaite minimiser. Un exemple de matrice est donné dans le tableau 2.2.

L'autre point de vue est lié au concept de score. Les valeurs utilisées reflètent la similarité entre les différents nucléotides. Les valeurs sur la diagonale de la matrice doivent être supérieures aux autres. La valeur associée à un couple de nucléotides est d'autant plus importante qu'ils sont similaires. La similarité maximale est par conséquent obtenue lorsque le score est le plus élevé. Il s'agit alors d'un problème de maximisation. L'utilisation d'un score est différente d'une distance, il peut donc ici y avoir des valeurs négatives. Nous proposons un exemple de matrice dans le tableau 2.3.

Les matrices de substitution pour les protéines

Pour les protéines, les matrices de substitutions ont été beaucoup plus étudiées. On peut trouver principalement 3 grandes familles, mais rien n'empêche d'utiliser une autre matrice si on le souhaite. Comme pour l'ADN, les matrices de substitution utilisées pour les protéines sont symétriques. En revanche toutes les matrices traditionnellement utilisées correspondent à des problèmes de maximisation. Les valeurs peuvent aussi bien être positives que négatives. Une affinité entre deux acides-aminés étant généralement caractérisée par une valeur positive.

les matrices PAM : la famille de matrices PAM [Dayhoff *et al.*, 1978] 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 39 (Distance PAM) *Soient S_1 et S_2 deux séquences protéiques. On dit*

qu'elles sont à une distance de 1 PAM (Point Accepted Mutation) 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.

Remarque 4 *Il est important de remarquer qu'une position donnée de la séquence peut avoir plusieurs mutations successives. La distance ne montre donc pas le nombre de points de divergences entre les deux séquences.*

Définition 40 (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. À titre d'exemple, la matrice *PAM 250* donnée au tableau 2.4 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. Nous pouvons constater que les valeurs associées à quelques acides aminés, A, N et S par exemple, sont assez faibles en valeur absolue car ils sont en effet assez fréquents. Inversement, le Tryptophane W est assez rare, et sa structure particulière empêche sa mutation en un autre acide aminé. La seule valeur positive du Tryptophane est obtenue lorsqu'il est apparié avec un autre W. Dans ce cas, la valeur associée est égale à 17.

Les matrices *PAM* ne sont pas les seules permettant d'attribuer une valeur à une paire de résidus.

Les matrices *Blosum* : les matrices *Blosum* [Henikoff and Henikoff, 1992] 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. Aussi, les calculs effectués pour les matrices de la famille *Blosum* ont ils été faits à partir de séquences similaires plus distantes.

La base de données *BLOCKS* [Henikoff and Henikoff, 1991] a été utilisée pour les calculs. 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 dûes à des mutations d'acides aminés.

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

Tab. 2.4 – La matrice PAM 250.

La première étape du calcul consiste à dénombrer dans une matrice toutes les paires de résidus contenues dans les blocs. On obtient ainsi la fréquence d'apparition de toutes les paires possibles d'acides aminés q_{ij} avec i et j appartenant à [1..20].

La fréquence d'apparition de l'acide aminé i est donnée par

$$p_i = q_{ii} + \sum_{i \neq j} q_{ij}/2$$

.

$$e_{ij} = \begin{cases} p_i^2, & i = j \\ 2.p_i.p_j, & i \neq j \end{cases}$$

Si la fréquence pratique est inférieure à la fréquence théorique, la valeur doit être négative. On prend donc le logarithme du rapport entre la fréquence pratique et la fréquence théorique. La matrice se calcule avec les valeurs :

$$s_{ij} = 2. \log_2 \left(\frac{q_{ij}}{e_{ij}} \right)$$

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. Ainsi, les matrices *Blosum 45* et *Blosum 62* qui sont les plus utilisées, ont été obtenues avec de blocs contenant respectivement 45% et 62% d'identité d'acides aminés.

Les matrices *Gonnet* : Les matrices *Gonnet* [Cohen *et al.*, 1994] sont calculées à partir de la distance *PAM* entre séquences. Toutefois les distances entre acides aminés ont été modifiées en réalisant des alignements successifs de séquences. A chaque alignement la matrice de distances est modifiée pour prendre en compte le nouveau résultat. L'objectif de ces alignements est d'obtenir une évaluation qui soit en adéquation avec le meilleur alignement possible.

2.3.6 Evaluation des brèches

L'évaluation des brèches dans un alignement est très importante. Selon les valeurs attribuées, le nombre de brèches peut varier. Si elles ne sont pas assez pénalisantes, la fonction de score risque de favoriser les alignements qui en contiennent beaucoup. En particulier, en morcellant les séquences il est possible d'augmenter le nombre de correspondances. Inversement, si les brèches sont trop fortement évaluées, les alignements ne comportant pas assez de brèches sont favorisés, empêchant ainsi d'avoir toutes les correspondances.

Il existe principalement deux modélisations [Altschul, 1989a] 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.

Les brèches à coût constant

Ce modèle de calcul pour les brèches est le plus simple, puisqu'il 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. En effet, comme toutes les positions d'une brèche ont la même valeur, l'évaluation repose sur le même principe que pour les paires de résidus. Il suffit pour cela de faire la somme de toutes les valeurs sur la longueur de l'alignement. Le symbole '−' peut alors être considéré comme un acide aminé pour l'évaluation.

Les brèches à coût affine

Selon les biologistes, l'évaluation à coût constant même si elle est simple à mettre en œuvre a un défaut majeur. Elle n'est en effet pas représentative de la façon dont les choses se passent dans la réalité. D'un point de vue biologique, la création de la brèche est beaucoup plus pénalisante que son élongation. Il semble alors normal de prendre en compte cette remarque importante, et donc d'associer un coût différent suivant qu'il s'agit du début ou de l'extension de la brèche.

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 $gap(l) = K + h.(l-1)$. On trouve également cette fonction sous la forme $gap(l) = K' + h.l$ en posant $K' = K - h$.

L'évaluation d'un alignement au moyen de cette méthode ne peut se faire de la même façon qu'avec les brèches à coût constant. Il est possible de réaliser le calcul suivant deux méthodes :

- La première consiste à n'évaluer que les paires de résidus, puis à y ajouter la somme des valeurs de toutes les brèches.
- La seconde méthode consiste à évaluer l'ensemble de l'alignement en faisant la somme de toutes les positions. Cette méthode nécessite lorsque l'on doit évaluer une brèche de savoir si la position précédente était également une brèche ou non. En pratique, seule cette méthode est utilisée pour la construction de l'alignement, même si elle génère plus de calculs que dans le cas des brèches à coût constant.

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 un ratio K/h voisin de 10.

Cas particulier des brèches de début et fin d'alignement

Lorsqu'un alignement comporte une brèche au début et/ou à la fin d'une des deux séquences, l'évaluation de celle-ci peut être différente. Comment une telle brèche doit-elle être considérée, puisqu'elle n'a pas été créée naturellement, mais par un biologiste ou un programme. On peut dans ce cas de figure rencontrer deux argumentations différentes.

La première consiste à dire que les deux séquences doivent être alignées ainsi parce qu'elles constituent théoriquement des zones identiques dans des génomes différents. Si le début ou la fin ne sont pas alignés correctement, cela provient d'une insertion ou d'une deletion qui doit donc être prise en compte comme telle. Il est donc nécessaire de compter le coût de cette brèche.

La deuxième façon de considérer le problème est de considérer que s'il y a une brèche en début ou en fin d'alignement, c'est uniquement parce que la séquence a été coupée. Cette brèche représente la partie de la séquence qui a été artificiellement supprimée. Dans ce cas comme on ne peut pas savoir si l'alignement est correct, il est impossible de donner une évaluation de cette partie. La brèche est alors considérée comme ayant un coût nul.

2.3.7 Les fonctions d'évaluation pour alignements par paires

Les fonctions d'évaluation ont pour principal objectif de déterminer la qualité d'un alignement. Idéalement, la fonction d'évaluation doit permettre de comparer plusieurs résultats. 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. Cette méthode consiste à parcourir toutes les paires de résidus de l'alignement. Lorsque les deux acides aminés sont identiques on attribue la valeur 1, et 0 dans les autres cas. On obtient ainsi le nombre d'identités de l'alignement. Les brèches sont comptabilisées dans les autres cas, c'est-à-dire qu'il n'y a pas de pénalités particulières comme exposé dans la partie précédente.

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 métode simple à utiliser mais elle n'est pas toujours très représentative. 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 qu'il n'est pas nécessairement gênant que deux acides aminés différents soient en regard.

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

Le coût associé à une brèche dépend de la méthode utilisée pour modéliser ce coût. S'il s'agit d'une brèche de début ou de fin d'alignement, elle peut éventuellement ne pas être comptée.

2.4 Rappels de programmation dynamique

Maintenant que nous savons évaluer la qualité d'un alignement, il devient possible d'envisager la recherche du meilleur alignement. Or nous avons vu que pour deux séquences données, la recherche de tous les alignements possibles ne peut être envisagée de façon raisonnable. Toutefois, pour éviter cela, il existe une méthode efficace basée sur la programmation dynamique.

2.4.1 Position du problème : étude d'un exemple simple

La suite de Fibonacci est un très bon exemple pour comprendre le fonctionnement et l'intérêt de la programmation dynamique. Pour cela commençons par en rappeler la définition de cette suite :

Définition 42 (Suite de Fibonacci) *La suite de Fibonacci est définie récursivement à partir de la somme de ses deux termes précédents :*
- $\mathcal{F}_1 = \mathcal{F}_2 = 1$
- $\mathcal{F}_{n+2} = \mathcal{F}_{n+1} + \mathcal{F}_n, \forall n \in \mathbb{N}^*$

De nombreuses méthodes permettent de calculer la valeur de cette suite pour une valeur $n \in \mathbb{N}$ donnée.

Méthode 1 : une méthode naïve

La plus intuitive, et sans aucun doute la plus maladroite, consiste à remplacer récursivement dans la formule générale jusqu'à obtenir les valeurs $\mathcal{F}_1$ et $\mathcal{F}_2$. Nous pouvons par exemple pour calculer $\mathcal{F}_{10}$ procéder de la façon suivante :

$$\begin{aligned}
\mathcal{F}_{10} &= \mathcal{F}_9 + \mathcal{F}_8 \\
&= \mathcal{F}_8 + \mathcal{F}_7 + \mathcal{F}_7 + \mathcal{F}_6 \\
&= \mathcal{F}_7 + \mathcal{F}_6 + \mathcal{F}_6 + \mathcal{F}_5 + \mathcal{F}_6 + \mathcal{F}_5 + \mathcal{F}_5 + \mathcal{F}_4 \\
&= \dots \\
&= 34 \times \mathcal{F}_2 + 21 \times \mathcal{F}_1 \\
&= 55
\end{aligned}$$

Méthode 2 : une méthode mathématique

Cette méthode est donnée car elle permet de calculer facilement la complexité de la première méthode.

Avec une récurrence de niveau 2, il est bien sûr facile de calculer directement toute valeur de la suite. La résolution de l'équation caractérisique associée donne les deux racines suivantes :

$$\begin{cases} \phi = (1 + \sqrt{5})/2 \\ \phi' = (1 - \sqrt{5})/2 \end{cases}$$

D'où la formule, dite de *Binet*, donnant la valeur de $\mathcal{F}_n$:

$$\mathcal{F}_n = \frac{1}{\sqrt{5}}.(\phi^n + \phi'^n)$$

Avec la première méthode que nous avons donnée, le nombre d'additions à réaliser pour calculer $\mathcal{F}_n$ est $\mathcal{F}_n - 1$. Or lorsque n tend vers l'infini, $\mathcal{F}_n$ est équivalent à $\phi^n/\sqrt{5}$. C'est-à-dire que le nombre de calculs à réaliser est exponentiel.

L'algorithme naïf consistant à calculer $\mathcal{F}_n$ par cette méthode a donc une complexité en $O(n)$.

Méthode 3 : la programmation dynamique

Le principal défaut de la première méthode proposée, est qu'il est nécessaire de calculer de nombreuses fois les mêmes valeurs $\mathcal{F}_i$. Par exemple pour calculer $\mathcal{F}_{10}$, il faut calculer $\mathcal{F}_9$ et $\mathcal{F}_8$. Or pour calculer $\mathcal{F}_9$ il faut également calculer $\mathcal{F}_8$. Ainsi, plus le calcul progresse, plus il est nécessaire de calculer de nombreuses fois les mêmes valeurs.

Cette façon de réaliser le calcul n'est donc pas performante. Et il semble plus judicieux de ne calculer qu'une seule fois les valeurs qui sont nécessaires. Au lieu de chercher à calculer $\mathcal{F}_n$ à partir des sous problèmes $\mathcal{F}_{n-1}$ et $\mathcal{F}_{n-2}$, il est préférable de calculer progressivement les différentes valeurs. Ainsi connaissant $\mathcal{F}_1$ et $\mathcal{F}_2$ il est possible de calculer $\mathcal{F}_3$ puis $\mathcal{F}_4$, etc. jusqu'à la valeur $\mathcal{F}_n$ cherchée.

Chaque étape du calcul est uniquement la somme des deux sous-problèmes précédemment obtenus. Il s'agit donc d'un calcul pouvant être fait à chaque fois en temps constant.

L'algorithme permettant de calculer $\mathcal{F}_n$ a alors une complexité linéaire en $O(n)$. Cette méthode de résolution correspond au principe de la *programmation dynamique*. Elle peut être appliquée dans certains cas pour la résolution de problèmes combinatoires. Nous allons maintenant exposer les conditions nécessaires pour l'application de la programmation dynamique.

2.4.2 Principe de la programmation dynamique

La programmation dynamique est un principe souvent simple à mettre œuvre pour résoudre des problèmes complexes. Elle ne s'applique toutefois qu'à une certaine catégorie de problèmes, et il est nécessaire de vérifier certaines conditions pour qu'elle puisse être appliquée.

Définition 43 (Problème et sous-problème) *Soit $\mathcal{P}(n)$ un problème d'optimisation de taille n. On appelle sous-problème de $\mathcal{P}(n)$ tout problème $\mathcal{P}(i)$ avec $i < n$.*

Définition 44 (Principe d'optimalité) *On dit qu'un problème $\mathcal{P}(n)$ satisfait au principe d'optimalité lorsqu'une solution optimale peut être exprimée en fonction de solutions optimales de sous-problèmes.*

Définition 45 (Programmation dynamique) *Soit $\mathcal{P}(n)$ un problème satisfaisant au principe d'optimalité. On dit qu'un algorithme de résolution de $\mathcal{P}(n)$ est basé sur le principe de la programmation dynamique s'il utilise les deux étapes suivantes :*

- *$\mathcal{P}(n)$ est calculé récursivement en partant des problèmes de plus bas niveau,*
- *Une table est construite dynamiquement pour conserver tous les résultats intermédiaires obtenus.*

L'utilisation de la programmation dynamique suppose donc que pour les valeurs les plus faibles de n, $\mathcal{P}(n)$ soit connu ou facile à calculer.

En conservant tous les résultats intermédiaires dans une table on évite d'avoir à les recalculer de nombreuses fois. En effet, refaire ainsi les mêmes calculs à chaque itération est à l'origine de la complexité exponentielle des algorithmes "naïfs".

2.5 Alignement global

2.5.1 Généralités

L'alignement de deux séquences tel que nous l'avons défini précédemment peut être décliné en deux sous-problèmes : *l'alignement global* et *l'alignement local*. Nous commençons ici avec l'alignement global en exposant un algorithme efficace basé sur la programmation dynamique. Même si la complexité est faible, celle-ci peut s'avérer encore trop forte dans certains cas très fréquents. Nous exposerons un algorithme permettant de résoudre ce problème.

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.

Le sens de meilleur alignement est bien entendu lié à la fonction d'évaluation utilisée. Or nous avons vu qu'il était possible d'évaluer les brèches de différentes façons, mais également de compter ou non les brèches de début et fin d'alignement. Dans tous les cas le principe général reste le même, mais les algorithmes résultants sont toutefois assez différents.

2.5.2 Algorithme de Needleman-Wunsch

Principe général

L'algorithme de *Needleman-Wunsch* [Needleman and Wunsch, 1970] est basé sur le principe de la programmation dynamique. Vouloir utiliser la programmation dynamique est assez intuitif, car il est aisé de définir un sous-problème du problème d'alignement. De plus le problème principal peut être exprimé en fonction de ses sous-problèmes.

Pour exposer le principe nous supposerons qu'il s'agit d'un problème de maximisation. Le raisonnement permettant d'expliquer le principe de construction peut être mené par induction sur la longueur de chacune des deux séquences.

Soient S et T deux séquences de longueurs p et q à aligner. Nous définissons le problème $\mathcal{P}(i, j)$ de la façon suivante :

- le problème consistant à déterminer le meilleur alignement des sous-séquences $S[1..i]$ et $T[1..j]$ pour $i > 0$ et $j > 0$,
- le problème se réduit à aligner $S[1..i]$ avec une brèche pour $i > 0$ et $j = 0$,
- le problème se réduit à aligner $T[1..j]$ avec une brèche pour $i = 0$ et $j > 0$,
- le cas fortement dégénéré où $i = j = 0$ n'a que peu d'intérêt, puisqu'il consiste à ne rien aligner. Nous le donnons toutefois ici car il nous servira de cas d'arrêt.

L'objectif pour nous est d'aligner les séquences S et T en totalité, ce qui correspond à déterminer $\mathcal{P}(p, q)$. L'évaluation de $\mathcal{P}(i, j)$ correspond à la valeur associée à cet alignement, nous la noterons $\mathcal{V}(i, j)$.

Un sous-problème $\mathcal{P}(i', j')$ de $\mathcal{P}(i, j)$ consiste :

- à déterminer le meilleur alignement des sous-séquences $S[1..i']$ et $T[1..j']$ de $S[1..i]$ et $T[1..j]$ pour $i' > 0$ et $j' > 0$,
- à aligner $S[1..i']$ avec une brèche pour $i' > 0$ et $j' = 0$,
- à aligner $T[1..j']$ avec une brèche pour $i' = 0$ et $j' > 0$.

Autrement dit, $\mathcal{P}(i', j')$ est un sous-problème de $\mathcal{P}(i, j)$ si et seulement si ($i' < i$ et $j' \leq j$) ou ($i' \leq i$ et $j' < j$).

Cherchons à exprimer $\mathcal{P}(i, j)$ en fonction de ses sous-problèmes. Trois cas sont alors à envisager suivant que i et j sont strictement positifs ou non.

Premier cas $i = 0$ et $j \neq 0$: Si $i = 0$, nous sommes dans un cas dégénéré du problème où il est simple d'exprimer $\mathcal{P}(0, j)$ en fonction de ses sous-problèmes. En effet selon les cas, il est possible de déterminer $\mathcal{P}(0, j)$ directement ou à partir de $\mathcal{P}(0, j - 1)$. Si les brèches de début et fin d'alignement ne sont pas prises en compte, la valeur associée à $\mathcal{P}(0, j)$ est bien sûr 0. Sinon, plusieurs cas peuvent être envisagés :

- pour les brèches à coût constant, le passage de $\mathcal{P}(0, j-1)$ à $\mathcal{P}(0, j)$ se fait par une extension de brèche. Donc $\mathcal{V}(0, j) = \mathcal{V}(0, j-1) + \alpha$,
- pour les brèches à coût affine, deux cas sont possibles :
  - si $j = 1$, il s'agit d'une création de brèche, et $\mathcal{V}(0, j) = K$,
  - sinon, il s'agit d'une extention de brèche, et $\mathcal{V}(0, j) = \mathcal{V}(0, j-1) + h$
- si un autre modèle mathématique est utilisé pour évaluer les brèches, dont l'évaluation est donnée f en fonction de la longueur, $\mathcal{V}(0, j) = f(j)$.

Deuxième cas $i \neq 0$ et $j = 0$: Ce cas est totalement symétrique du précédent, et les résultats obtenus sont identiques en remplaçant j par i.

Troisième cas $i \neq 0$ et $j \neq 0$: Il s'agit du cas général, et nous allons exprimer $\mathcal{P}(i, j)$ en fonction de ses sous-problèmes. Pour déterminer $\mathcal{P}(i, j)$, 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,
- $T[i]$ 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 $T[1..j]$. Or il s'agit là du problème $\mathcal{P}(i-1, j)$, qui est un sous-problème de $\mathcal{P}(i, j)$. Ce qui veut dire que par hypothèse, $\mathcal{V}(i-1, j)$ est connue, et donc la valeur de l'alignement avec $S[i]$ aligné avec une brèche a pour valeur $\mathcal{V}(i-1, j) + val(S[i], '-')$, où val est la fonction définie à la définition 41.

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

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 $\mathcal{P}(i-1, j-1)$, qui est également un sous-problème de $\mathcal{P}(i, j)$. La valeur associée est donc donnée par $\mathcal{V}(i-1, j-1) + val(S[i], T[j])$.

Ces calculs effectués, il ne reste plus qu'à déterminer laquelle des trois solutions est optimale. Ainsi dans le cas d'une maximisation nous obtenons :

$$\mathcal{V}(i, j) = \max \begin{pmatrix} \mathcal{V}(i-1, j) + val(S[i], '-'), \\ \mathcal{V}(i, j-1) + val('-', T[j]), \\ \mathcal{V}(i-1, j-1) + val(S[i], T[j]) \end{pmatrix} \tag{2.1}$$

Pour une minimisation, il suffit de remplacer “max” par “min”.

La récursivité ainsi définie satisfait au *Principe d'optimalité* puisque l'optimum est exprimé en fonction d'optima de 3 sous-problèmes. La programmation dynamique peut donc être utilisée pour résoudre le problème. La fonction val est toutefois trompeuse puisqu'elle sous-entend que l'on sait évaluer le coût de la brèche alignée avec $S[i]$ ou $T[j]$. Lorsque les brèches sont à coût constant, c'est en effet le cas. En revanche pour les brèches

à coût affine, rien ne permet à la fonction *val* de savoir s'il s'agit d'une ouverture ou d'une extension de brèche. Il est donc nécessaire de différencier les deux cas, le premier étant beaucoup plus simple à résoudre.

Afin de déterminer $\mathcal{P}(p, q)$ 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 $\mathcal{P}(i, j)$ avec $0 \leq i \leq p$ et $0 \leq j \leq q$. Traditionnellement, la représentation utilisée est celle de la figure 2.2.

	-	C	T	G	G	A
-						
C						
A						
G						
A						

FIG. 2.2 – Exemple de tableau utilisé pour la programmation dynamique.

Cas des brèches à coût constant

Nous avons vu que dans ce cas, si i et j ne sont pas nuls, $\mathcal{P}(i, j)$ peut être déterminé à partir de ses trois sous-problèmes directs $\mathcal{P}(i-, j)$, $\mathcal{P}(i, j - 1)$ et $\mathcal{P}(i - 1, j - 1)$. Par rapport à la représentation montrée à la figure 2.2, cela correspond à la représentation de la figure 2.3.

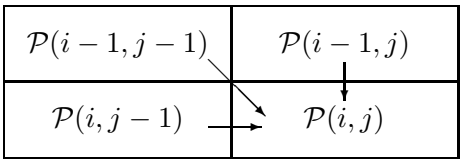

FIG. 2.3 – Calcul de $\mathcal{P}(i, j)$ à partir de ses sous-problèmes directs.

Le tableau va donc permettre de conserver les valeurs $\mathcal{V}(i, j)$, et progressivement d'aboutir au calcul de $\mathcal{V}(p, q)$, valeur de l'alignement optimum. Pour cela, il suffit d'utiliser les équations vues précédemment pour les problèmes de plus bas niveau, c'est-à-dire pour $i = 0$ ou $j = 0$. Comme nous l'avons vu les équations sont

- $\mathcal{V}(i, 0) = \mathcal{V}(i - 1, 0) + \alpha$
- $\mathcal{V}(0, j) = \mathcal{V}(0, j - 1) + \alpha$

Dans le cas où les brèches de début d'alignement ne sont pas comptées, $\alpha = 0$ et donc $\mathcal{V}(i, 0) = \mathcal{V}(0, j) = 0$.

La première ligne et la première colonne du tableau peuvent ainsi être calculées. Ensuite, il faut appliquer récursivement l'équation 2.1 dans le tableau de valeur. Le calcul

peut être fait lorsque les valeurs des cases situées “à gauche”, “au dessus” et en “diagonale” sont connues. Le principe calculatoire généralement utilisé consiste à effectuer tous les calculs ligne par ligne ou colonne par colonne jusqu'à la dernière case du tableau.

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

$\mathcal{M}$	$-$	A	C	G	T
$-$	/	1	1	1	1
A	1	6	3	3	3
C	1	3	6	3	3
G	1	3	3	6	3
T	1	3	3	3	6

Les brèches sont à coût constant 1, donc pour la première ligne et la première colonne nous avons :

- $\mathcal{V}(i,0) = i$
- $\mathcal{V}(0,j) = j$

Le résultat de cette première étape est donné dans la figure 2.4.

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

FIG. 2.4 – Calcul des valeurs associées aux problèmes de plus bas niveau.

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

$$\mathcal{V}(1,1) = \max \begin{pmatrix} \mathcal{V}(0,1) + val(C,-), \\ \mathcal{V}(1,0) + val(-,C), \\ \mathcal{V}(0,0) + val(C,C) \end{pmatrix} \tag{2.2}$$

$$\mathcal{V}(1,1) = \max \begin{pmatrix} 1+1, \\ 1+1, \\ 0+6 \end{pmatrix} \tag{2.3}$$

$$\mathcal{V}(1,1) = 6 \tag{2.4}$$

En faisant le calcul par ligne, il devient possible de calculer $\mathcal{V}(1,2)$.

$$\mathcal{V}(1,2) = \max \begin{pmatrix} 6+1, \\ 2+1, \\ 1+3 \end{pmatrix} = 7 \tag{2.5}$$

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

	-	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	17	22

FIG. 2.5 – Résultat complet.

Construction de l'alignement

La méthode que nous venons de présenter permet de déterminer la valeur optimale $\mathcal{V}(p, q)$ 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 constuire 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 $\mathcal{V}(p, q)$. Puisque l'on sait qu'il s'agit de l'optimum obtenu à partir des trois sous-problèmes $\mathcal{P}(p-1, q)$, $\mathcal{P}(p, q-1)$ et $\mathcal{P}(p-1, q-1)$. Suivant le sous-problème ayant permis de calculer $\mathcal{V}(p, q)$, trois cas sont envisageables :

- Le cas $\mathcal{P}(p-1, q)$ correspond à une insertion dans S,
- Le cas $\mathcal{P}(p, q-1)$ correspond à une deletion dans T,
- Le cas $\mathcal{P}(p-1, q-1)$ 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 $\mathcal{P}(0, 0)$, 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 $\mathcal{P}(i, j)$ 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 de la figure 2.6

	-	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	17	22

FIG. 2.6 – Indication à chaque étape des sous-problèmes utilisés.

On en déduit donc les alignements suivants pour un coût de 2 :

```
CTGGA      CTGGA      CTGGA
CA-GA      CAG-A      C-AGA
```

Cas des brèches à coût affine

De même que pour l'alignement avec brèches à coût constant, il est possible dans ce cas d'utiliser un algorithme basé sur le principe de la programmation dynamique [Gotoh, 1982]. L'utilisation d'un coût affine pour évaluer les brèches nécessite de savoir s'il s'agit d'une création ou d'une extension de brèche lorsque l'on fait une insertion ou une deletion. Il n'est donc plus possible d'exprimer directement $\mathcal{P}(i,j)$ en fonction de ses trois sous-problèmes directs.

Même s'il est nécessaire de prendre en compte plus de sous-problèmes, le principe reste le même. La notation utilisée précédemment ne peut plus être utilisée directement ; pour le montrer prenons par exemple le cas de l'insertion. Avec l'utilisation des brèches à coût constant, nous avions $\mathcal{V}(i,j) = \mathcal{V}(i-1,j) + cste$ pour l'insertion. Avec les brèches à coût affine, il est nécessaire de pouvoir différencier si l'opération précédente était également une insertion ou non. Il nous faut donc connaître trois valeurs différentes correspondant aux trois alignements des sous-séquences $S[1..i-1]$ et $T[1..j]$ suivants :

- Le meilleur alignement se terminant par une insertion,
- Le meilleur alignement se terminant par une deletion,
- Le meilleur alignement se terminant par un match ou un mismatch.

Par rapport au cas des brèches à coût constant, il est donc nécessaire de connaître pour chaque sous-problème direct les trois valeurs possibles suivant la dernière opération d'édition qui y est effectuée. Il nous faut donc décomposer chaque sous-problème direct $\mathcal{P}(i,j)$, et nous utiliserons les notations suivantes :

Mise en équations du problème

Soit $DM(i,j)$ la valeur maximale que l'on peut obtenir pour $\mathcal{V}(i,j)$ si la dernière opération est issue de la diagonale, c'est-à-dire, s'il s'agit d'une opération de type match / mismatch. On définit de même les valeurs $HM(i,j)$ et $VM(i,j)$, où $HM(i,j)$ est la valeur maximale que peut avoir $\mathcal{V}(i,j)$ si la dernière opération est une insertion (c'est-à-dire que la dernière opération est obtenue horizontalement), et $VM(i,j)$ est la valeur maximale que peut avoir $\mathcal{V}(i,j)$ si la dernière opération est une deletion (opération obtenue verticalement). On obtient par conséquent :

$$\mathcal{V}(i,j) = \max\{DM(i,j), HM(i,j), VM(i,j)\}$$

Le calcul de $HM(i,j)$ et $VM(i,j)$ nécessite de savoir si des brèches ont été préalablement insérées, c'est-à-dire de savoir si l'opération précédente pour $HM(i,j)$ était une insertion, et si l'opération précédente pour $VM(i,j)$ était une deletion. Il faut donc pouvoir conserver à chaque étape la valeur maximale obtenue par l'opération de match/mismatch, par l'opération d'insertion et par l'opération de deletion.

Pour $HM(i,j)$ le calcul s'effectue en faisant une opération d'insertion à partir des sous-problèmes $DM(i,j-1)$, $HM(i,j-1)$ et $VM(i,j-1)$. Pour la valeur issue du sous-problème $HM(i,j-1)$ le coût d'insertion de la brèche est h, alors que pour les deux autres cas, l'insertion aura un coût de K. En appelant $HD(i,j)$, $HH(i,j)$ et $VH(i,j)$ les valeurs obtenues pour ses trois sous-problèmes, on obtient :

$$\begin{cases} HD(i,j) = DM(i,j-1) + K \\ HH(i,j) = HM(i,j-1) + h \\ HV(i,j) = VM(i,j-1) + K \end{cases}$$

La valeur de $HM(i,j)$ est donc obtenue en prenant le maximum des trois :

$$HM(i,j) = \max\{HD(i,j), HH(i,j), HV(i,j)\}$$

On calcule de la même façon la valeur de $VM(i,j)$ par une opération de deletion à partir de $DM(i-1,j)$, $HM(i-1,j)$ et $VM(i-1,j)$. La deletion a un coût de h pour le calcul à partir du sous-problème $VM(i-1,j)$, et un coût de K pour les autres sous-problèmes. En appelant $VD(i,j)$, $VH(i,j)$ et $VV(i,j)$ les valeurs obtenues pour ses trois sous-problèmes, on obtient :

$$\begin{cases} VD(i,j) = DM(i-1,j) + K \\ VH(i,j) = HM(i-1,j) + K \\ VV(i,j) = VM(i-1,j) + h \end{cases}$$

La valeur de $VM(i,j)$ est donc obtenue en prenant le maximum des trois :

$$VM(i,j) = \max\{VD(i,j), VH(i,j), VV(i,j)\}$$

Le calcul de $DM(i,j)$ s'obtient à partir des trois sous-problèmes $DM(i-1,j-1)$, $HM(i-1,j-1)$ et $VM(i-1,j-1)$. Soit $\mathcal{M}$ la matrice de substitution utilisée pour l'alignement des deux séquences, et soit $S(i)$ le $i^{\text{ème}}$ caractère de la première séquence et $T(j)$ le $j^{\text{ème}}$

caractère de la seconde séquence. On obtient $DM(i,j)$ en ajoutant $\mathcal{M}(S(i),T(j))$ à chacun des sous-problèmes :

$$\begin{cases} DD(i,j) = DM(i-1,j-1) + \mathcal{M}(S(i),T(j)) \\ DH(i,j) = HM(i-1,j-1) + \mathcal{M}(S(i),T(j)) \\ DV(i,j) = VM(i-1,j-1) + \mathcal{M}(S(i),T(j)) \end{cases}$$

La valeur de $DM(i,j)$ est donc obtenue en prenant le maximum des trois :

$$DM(i,j) = \max\{DD(i,j), DH(i,j), DV(i,j)\}$$

La valeur de $\mathcal{V}(i,j)$ est donc donnée par :

$$M(i,j) = \max \begin{cases} DM(i,j) = \max \begin{cases} DD(i,j) = DM(i-1,j-1) + \mathcal{M}(S(i),T(j)) \\ DH(i,j) = HM(i-1,j-1) + \mathcal{M}(S(i),T(j)) \\ DV(i,j) = VM(i-1,j-1) + \mathcal{M}(S(i),T(j)) \end{cases} \\ HM(i,j) = \max \begin{cases} HD(i,j) = DM(i,j-1) + K \\ HH(i,j) = HM(i,j-1) + h \\ HV(i,j) = VM(i,j-1) + K \end{cases} \\ VM(i,j) = \max \begin{cases} VD(i,j) = DM(i-1,j) + K \\ VH(i,j) = HM(i-1,j) + K \\ VV(i,j) = VM(i-1,j) + h \end{cases} \end{cases}$$

Cette formule est valable pour $i > 0$ et $j > 0$.

Le principe du calcul de $\mathcal{V}(i,j)$, valeur maximale de l'alignement des i premiers caractères de S avec les j premiers caractères de T peut être représenté de façon simplifiée sous la forme d'un graphique (Fig. 2.7).

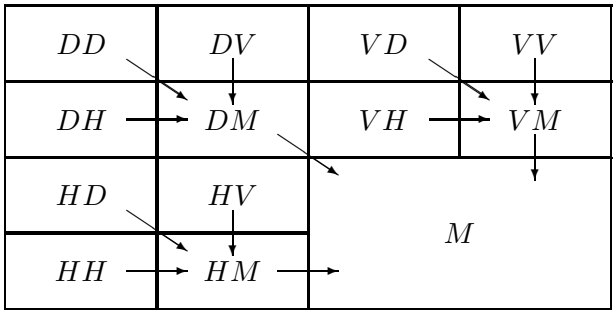

Fig. 2.7 – Principe du calcul de $\mathcal{V}(i,j)$ à partir de $DM(i,j)$, $HM(i,j)$ et $VM(i,j)$.

Les cas de base

Les cas de base correspondent à $i = 0$ ou à $j = 0$. Dans les cas où elles existent, les valeurs associées peuvent être calculées directement :

- Initialisation pour $i = 0$ et $j = 0$. Il s'agit du cas où aucun caractère n'a encore été aligné ni dans la séquence S ni dans la séquence T. La valeur associée pour toutes les matrices précédemment citées est donc 0.
- Initialisation pour $i > 0$ et $j = 0$. Il s'agit des valeurs présentes dans la première colonne des matrices. Elles correspondent à l'insertion d'une brèche de longueur i au début de la séquence S. Pour plusieurs matrices, cette opération n'a pas de signification. Parmi les matrices DD, DH, DV, HD, HH, HV, VD, VH et VV le calcul n'est défini que pour la matrice VV. En effet, cette matrice est celle qui prend en compte le calcul pour l'insertion dans S. Parmi les matrices DM, HM et VM, seule VM est donc définie et égale à VV. M est donc calculée uniquement avec VM. Pour $i = 1$ la valeur de VV correspond au coût d'ouverture d'une brèche, pour $i > 1$, $VV(i, 0) = VM(i - 1, 0)+$ le coût d'insertion.

$$VV(i,0) = VM(i,0) = M(i,0) = \begin{cases} K, \, i = 1 \\ VM(i-1,0) + h, \, \forall i > 1 \end{cases}$$

- Initialisation pour $i = 0$ et $j > 0$. Il s'agit des valeurs présentes dans la première ligne des matrices. Elles correspondent à l'insertion d'une brèche de longueur j au début de la séquence T. Comme précédemment, cette opération n'est définie que pour les matrices HH, HM et M. Pour $j = 1$, il s'agit également du coût d'ouverture d'une brèche, pour $j > 1$ de l'ajout du coût d'extension de la brèche.

$$HH(0,j) = HM(0,j) = M(0,j) = \begin{cases} K, \, j = 1 \\ HM(0,j-1) + h, \, \forall j > 1 \end{cases}$$

La présence de valeurs non définies dans les matrices ne permet pas d'utiliser les formules de récurrence que nous avons vues. D'un point de vue pratique, toutes les valeurs non définies sont donc initialisées à $-\infty$. Lors du calcul de la première ligne et de la première colonne, VM ou HM sont calculées, et donc intialisées avec des valeurs différentes de $-\infty$. A partir de la deuxième ligne et de la deuxième colonne DM, HM et VM sont calculés avec la formule de récurrence définie plus haut.

Par exemple pour HM, on a

$$HM(i,j) = \max\{DM(i,j-1) + K, HM(i,j-1) + h, VM(i,j-1) + K\}$$

Or nous venons de montrer que $HM(i, j - 1)$ ou $VM(i, j - 1)$ est différent de $-\infty$, donc $HM(i, j)$ est également différent de $-\infty$. On montre de même que $DM(i, j)$ et $VM(i, j)$ sont aussi tous deux définis, et par conséquent $\mathcal{V}(i, j)$ est également définie.

Par récurrence, on peut donc montrer que si les valeurs de DM, HM et VM sont définies pour la colonne c, $c \geq 0$ et la ligne l, $l \geq 0$, alors, les valeurs de DM, HM et VM sont également définies pour la colonne $c + 1$ et la ligne $l + 1$.

Construction de l'alignement

Le principe de construction de l'alignement est assez semblable à celui des brèches à coût constant. Il faut chercher de proche en proche quels sous-problèmes ont permis de mener au résultat. L'alignement est là encore construit à partir de la fin en partant du résultat. Toutefois, la méthode de calcul différente conduit à un parcours différent des matrices pour construire l'alignement. Il est en effet nécessaire à chaque étape de prendre en compte à partir de quelle matrice sera faite l'étape suivante de la construction.

Complexité de l'alignement de deux séquences

Pour les sous-problèmes de plus bas niveau, le calcul peut être fait directement par une simple addition. Chaque case de la première ligne et de la première colonne peut donc être calculée en temps constant. Le calcul de $\mathcal{P}(i,j)$ dans le cas général consiste à déterminer le maximum de plusieurs valeurs. Chacune de ces valeurs pouvant également être déterminée en temps constant. Comme pour les sous-problèmes de plus bas niveau, le calcul de $\mathcal{V}(i,j)$ peut donc être fait en temps constant.

Seule la première case du tableau ne nécessite pas de calcul. Le nombre de cases du tableau à calculer est donc $(p+1)\times(q+1)-1$. On obtient donc une complexité en $O(p.q)$ pour le calcul de la valeur de $\mathcal{P}(p,q)$.

Regardons maintenant la complexité pour construire un alignement des deux séquences. Le pire des cas pour la construction correspond à un alignement de taille maximum. Ce cas se produit lorsque les deux séquences sont totalement disjointes, et donc avec un alignement de longueur $p+q$. Chaque étape de la construction se fait en temps constant, la construction d'un alignement a donc une complexité en $O(p+q)$. Le calcul complet d'un alignement de deux séquences a donc une complexité en $O(p.q)$.

Lorsqu'il y a plusieurs alignements possibles pour deux séquences, chaque construction a une complexité en $O(p+q)$. Or il n'est pas possible de déterminer à l'avance le nombre d'alignements optimaux différents pour deux séquences données.

2.6 Alignement local

2.6.1 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.

Définition 46 (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 $Eval(B) > Eval(A)$ (ou $Eval(B) < Eval(A)$ dans le cas d'une minimisation), Eval étant la fonction définie à la définition 41.*

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.

2.6.2 Algorithme de *Smith-Waterman*

Définition

L'algorithme de *Smith et Waterman* [Smith and Waterman, 1981] 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 $\mathcal{V}$ 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 2.1 devient :

$$\mathcal{V}(i,j) = \max \begin{pmatrix} \mathcal{V}(i-1,j) + val(S[i],'-'), \\ \mathcal{V}(i,j-1) + val('-',T[j]), \\ \mathcal{V}(i-1,j-1) + val(S[i],T[j]), \\ 0 \end{pmatrix} \tag{2.6}$$

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 l'équation 2.6,
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 5 *:*

– *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 2.6 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. On peut 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`

Les valeurs de la matrice de substitution utilisées pour déterminer l'alignement sont les suivantes :

- match : 5,
- mismatch : -4,
- insertion ou deletion : -7.

Le détail des calculs ainsi que le chemin suivi pour la construction sont donnés dans le tableau 2.5. La valeur maximale calculée est 19. Elle constitue donc le point de départ pour

	-	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

FIG. 2.8 – *Exemple d'alignement local*

la construction de l'alignement local. En remontant jusqu'à la dernière valeur strictement positive on obtient le résultat suivant :

```
GAAG-GCA
GCAGAGCA
```

2.6.3 Résultats et complexité

L'algorithme d'alignement local est très similaire à l'algorithme d'alignement global. L'ajout de la valeur 0 dans le calcul de $\mathcal{V}(i, j)$ ne modifie pas la complexité du calcul car il s'agit maintenant de trouver le maximum de 4 valeurs et non plus 3. La complexité pour déterminer la matrice entière est donc la même que celle de l'alignement global : $O(p.q)$.

Pour la construction de l'alignement résultat, on utilise également la même méthode. La seule différence est d'adapter l'algorithme à la partie qui doit être construite. Il faut

donc tout d'abord déterminer où commence la construction, pour cela il faut trouver la valeur maximale dans la matrice. Pour ne pas augmenter la complexité de cette partie de l'algorithme, il est préférable de conserver cette valeur lors du calcul de la matrice. Un alignement de longueur maximale représente le pire des cas pour la construction du résultat. Or nous avons vu précédemment que cette complexité est en $O(p + q)$. S'il y a plusieurs occurrences de la valeur maximale dans la matrice, la complexité de chaque alignement est également en $O(p + q)$.

La complexité de l'algorithme pour obtenir un alignement local est finalement en $O(p.q)$, c'est-à-dire la même que pour un alignement global.

2.6.4 *Blast* : Basic Local Alignment Search Tool

Introduction

Nous avons vu que l'alignement de deux séquences est fréquemment utilisé pour déterminer la similarité. En particulier cette méthode est employée pour les recherches dans une base de séquences. Ainsi, il faut pouvoir déterminer à partir d'une séquence toutes les séquences de la base de données qui lui sont similaires.

Le problème semble facile à résoudre puisque nous avons vu un algorithme simple permettant de déterminer l'alignement de deux séquences. De plus, une fois l'alignement obtenu, il suffit de le parcourir pour déterminer la similarité entre les deux séquences, opération qui peut être faite en temps linéaire. La complexité pour comparer deux séquences de longueurs p et q est donc égale à la complexité de l'alignement, c'est-à-dire $O(p.q)$.

Même si cette complexité est faible, elle est pourtant trop importante pour que l'algorithme d'alignement global puisse être utilisé directement. En effet, les bases de données contiennent bien souvent plusieurs millions de séquences, et réaliser tous les alignements prend beaucoup trop de temps.

Présentation de l'algorithme

Blast [Altschul *et al.*, 1990] a donc été créé pour apporter une solution à ce problème.

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 3 pour les protéines. Chacune des sous-séquences est comparée aux sous-séquences de taille w des séquences de la base 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 base 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.

Remarques

- Les implémentations faites de l'algorithme permettent d'obtenir les résultats beaucoup plus rapidement. Les temps de calculs sont en effet divisés par 50.
- Cet algorithme étant de loin le plus utilisé dans ce domaine, il existe de nombreuses implémentations. Chacune étant spécifique à un type de problème.
- L'algorithme peut être téléchargé gratuitement, ou être utilisé en ligne via un navigateur.
- Citons pour terminer *Fasta* [Pearson and Lipman, 1988], un autre algorithme également destiné à réaliser des requêtes de similarité d'une séquence sur une base de données.

2.7 Conclusion

Dans ce chapitre nous avons présenté un problème important de la biologie moléculaire : le problème d'alignement de deux séquences. Il s'agit d'un problème très fréquent en bioinformatique parce qu'il a des utilisations pratiques mais également parce qu'il va être utilisé pour d'autres opérations. En effet, comme nous le verrons au prochain chapitre, il constitue bien souvent une étape nécessaire pour de nombreux algorithmes d'alignement multiple.

Le nombre d'alignements possibles de deux séquences est très important. Il est donc nécessaire de définir des critères d'évaluation afin de déterminer quel est le meilleur alignement en fonction de ces critères. On utilise pour cela une matrice de substitution qui à chaque paire de résidus va associer une valeur, ainsi qu'une fonction permettant d'évaluer le coût d'insertion des brèches. Pour cela il existe principalement deux modélisations : l'utilisation d'un coût constant et l'utilisation d'un coût affine.

À partir des critères d'évaluation retenus, tout alignement de deux séquences peut être évalué. Les opérations d'édition sont utilisées dans cet alignement pour indiquer les transformations permettant de passer de la première séquence à la deuxième. L'évaluation d'un alignement est donc égale à la somme de toutes ces opérations d'édition. Toutefois, le nombre d'alignements possibles ne permet pas de les chercher tous afin de déterminer lequel est le meilleur.

Le principe des opérations d'édition est très utile car il permet de déterminer le meilleur alignement de deux séquences dès lors que l'on connaît les meilleurs alignements des sous-problèmes directs. En remontant de proche en proche les sous-problèmes directs, on aboutit à des sous-problèmes de bas niveau représentant des cas dégénérés d'alignements. Ces sous-problèmes peuvent être facilement résolus, il est donc possible de calculer tous les sous-problèmes. On peut donc obtenir le meilleur alignement de deux séquences en utilisant un algorithme basé sur le principe de la programmation dynamique. Pour deux séquences de longueurs p et q la complexité de cet algorithme est en $0(p.q)$. Un algorithme ayant la même complexité permet de déterminer le meilleur alignement local des deux séquences.

Chapitre 3

Alignement multiple de séquences

Après avoir vu le cas particulier de l'alignement de deux séquences, nous généralisons le problème à un nombre quelconque de séquences. Dans un premier temps nous allons présenter le problème, et nous verrons que la forte complexité ne permet pas en pratique d'utiliser des méthodes exactes. Le problème d'alignement multiple de séquences étant très important, de nombreux algorithmes ont été développés. Nous présenterons donc une classification des méthodes, ainsi que les plus réprésentatives dans chacune des catégories.

Souvent pour un algorithme donné, la qualité des résultats dépend du type de séquences devant être alignées. Il est donc important de pouvoir choisir une méthode adaptée au problème, nous terminerons donc ce chapitre en présentant des bases de jeux d'essais, ainsi que les critères permettant de comparer les résultats fournis par les différents algorithmes.

3.1 Généralités

Dans cette partie nous présentons le problème d'alignement multiple de séquences. Pour un jeu de séquences donné, le nombre d'alignements multiples est très important, nous verrons comment il est possible d'évaluer leur qualité. L'algorithme basé sur la programmation dynamique que nous avons vu au chapitre précédent peut être généralisé, mais nous verrons que la complexité du problème ne permet pas de l'utiliser en pratique.

3.1.1 Définitions

La problème vu au chapitre précédent peut être généralisé pour un ensemble de k séquences, avec $k > 2$. On parle alors *d'alignement multiple de séquences*. Les définitions ainsi que les propriétés vont également pouvoir être généralisées.

Définition 47 (Alignement Multiple de Séquences) *Soit $S = \{S_1, S_2, \ldots, S_k\}$ 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 $S' = \{S'_1, S'_2, \ldots, S'_k\}$ 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.*

Il est possible de définir de façon équivalente le problème d'alignement multiple de séquences au moyen d'une matrice $k \times n$. 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 "$-$".

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

3.1.2 Evaluation d'un alignement multiple

Le critère d'évaluation utilisé est souvent une fonction de score. Soit f une fonction permettant de définir la qualité d'un alignement multiple. f est une fonction définie sur Σ'^2 et à valeurs dans $\mathbb{R}$. Nous donnons dans la section suivante quelques exemples des fonctions les plus utilisées.

Le nombre d'alignements multiples possibles pour l'ensemble des séquences de S est très important. Il est donc nécessaire de pouvoir évaluer la qualité de chaque solution. Quelques fonctions ont été définies afin de permettre une évaluation des alignements, mais elles peuvent également être utilisées par des algorithmes lors de la construction de ces alignements.

3.2 Les fonctions d'évaluation

3.2.1 La somme des paires

Nous avons donné au chapitre précédent une fonction permettant d'évaluer un alignement de deux séquences. De même que nous avons généralisé la définition d'alignement de deux séquences à un alignement de $k > 2$ séquences, cette fonction peut également être généralisée.

Définition 48 (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 :*

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

Cette définition de Som dépend de la fonction f permettant d'évaluer une paire de résidus. f est donc définie sur $\Sigma' \times \Sigma'$ et est à valeurs dans $\mathbb{R}$. La fonction dépend donc de la matrice de substitution utilisée, mais elle dépend également de la méthode utilisée pour évaluer les brèches. Même si en pratique les brèches sont très souvent évaluées avec la méthode de coût affine, il est également possible d'utiliser un coût constant.

On constate aisément qu'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 :

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

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

L'avantage de la fonction de somme des paires est qu'elle permet d'évaluer les zones correspondant à un bon alignement en leur donnant des valeurs importantes. En effet, la forme même de la fonction permet de surévaluer les colonnes contenant de nombreux appariements. Pour une colonne contenant deux occurrences de la même lettre, soit v la valeur indiquée dans la matrice de substitution $\mathcal{M}$. Si dans une colonne il y a α occurrences de cette même lettre, la valeur associée est égale à $\alpha.(\alpha-1).v/2$. La valeur relative d'une colonne contenant peu d'appariements est donc négligeable devant une colonne qui en contient beaucoup.

3.2.2 La somme des paires pondérées

La fonction de somme de paires que nous venons de présenter est simple à mettre en œuvre, et elle permet d'évaluer la qualité d'un alignement multiple. La raison principale de la pertinence de cette fonction est liée au terme en α^2. En effet lorsque les séquences alignées sont très similaires, la fonction donne des valeurs très élevées lorsque l'alignement est de bonne qualité.

Pourtant sous cette forme, la fonction peut avoir un défaut important. Lorsque toutes les séquences sont très similaires, la fonction est tout à fait adaptée. En revanche, lorsque ce n'est pas le cas cela peut poser des problèmes [Vingron and Argos, 1989], [Altschul *et al.*, 1989]. Un exemple fréquent que l'on peut trouver dans les jeux de séquences est celui des séquences dites *orphelines*. Lorsque sur un ensemble de séquences à aligner, toutes sauf une sont similaires, la séquence différente est dite orpheline. Sa valeur au sein de l'alignement est négligeable par rapport aux autres séquences, la façon dont cette séquence est alignée n'a que très peu d'influence sur le résultat que donne la fonction. On peut remédier à ce problème en multipliant par un coefficient supérieur à un toutes les évaluations associées à cette séquence.

L'exemple de la séquence orpheline est très représentatif, mais d'une façon plus générale 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. Le principe est assez simple : pondérer les séquences pour pouvoir donner plus d'importance à celles qui sont les plus distantes. Ainsi, il devient possible de pondérer chaque alignement par paire de la somme des paires, en donnant d'autant plus de poids que les séquences sont distantes des autres.

Définition 49 (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 :*

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

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 [Saitou and Nei, 1987]. Le principe est assez simple, et sera exposé plus avant dans le chapitre. Cet algorithme est en effet utilisé lors de la construction d'un *guide-tree* et nous verrons comment calculer le poids de chaque séquence.

3.2.3 La fonction *Coffee*

L'algorithme *Coffee* [Notredame *et al.*, 1998] propose une méthode différente pour évaluer les alignements multiples. La méthode utilise ici aussi les alignements par paires, mais pas de la même façon. La fonction d'évaluation prend en compte le fait que l'algorithme permettant de déterminer le meilleur alignement de deux séquences est un algorithme exact.

Soit $S = \{S_1, S_2, \ldots, S_k\}$, un ensemble de $k > 2$ séquences, et soit $A = \{S'_1, S'_2, \ldots, S'_k\}$ 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 :

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

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 :

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

où *Long* est la fonction qui a un alignement de séquences associe 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 $SCORE(A_{ij}) = Long(A_{ij})$. Par conséquent, la fonction peut se simplifier, et on obtient Coffee$(A) = 1$.

À l'opposé, si aucune paire de résidus de A_{ij} n'est présente dans l'alignement S_{ij} quels que soient i et j, alors le numérateur de la fonction Coffee est toujours nul, et donc 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.

Le principal avantage de la fonction *Coffee* est qu'elle offre une méthode d'évaluation qui est plus proche du problème lui-même. La matrice de substitution et la fonction d'évaluation ne sont utilisées que pour réaliser les alignements par paires.

3.2.4 Les fonctions de *T-Coffee*, *M-Coffee* et *3D-Coffee*

La fonction *Coffee* que nous venons de présenter est calculée à partir des alignements par paires, car ceux-ci sont obtenus par un algorithme exact. Les fonctions des algorithmes *T-Coffee* [Notredame *et al.*, 2000], *M-Coffee* [Wallace *et al.*, 2006] et *3D-Coffee* [O'Sullivan *et al.*, 2004] reprennent ce principe, à savoir, évaluer un alignement multiple A des séquences de S à partir d'alignements d'une partie de S.

La fonction de *T-Coffee*

La fonction utilisée dans l'algorithme *T-Coffee* reprend le principe que nous venons de voir, mais en l'étendant à d'autres alignements. 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, mais également les alignements de tous les triplets de séquences. Comme nous le verrons dans ce chapitre, il est en effet possible d'aligner 3 séquences de façon exacte.

Les résultats ainsi obtenus permettent de constituer une bibliothèque d'alignements pour l'évaluation de A. Cette bibliothèque va permettre d'assigner une valeur à chaque paire de résidus. Plus cette paire de résidus est présente dans de nombreux alignements, et plus une valeur élevée lui est attribuée.

Comme pour *Coffee*, cette fonction offre l'avantage de s'adapter à chaque problème. Elle permet toutefois de corriger un défaut de *Coffee*, où pour deux séquences S_i et S_j les paires de résidus sont toujours uniques. C'est-à-dire qu'à une position donnée de S_i ne peut correspondre qu'une et une seule postion de S_j. Avec *T-Coffee*, un résidu de S_i peut éventuellement être aligné avec plusieurs résidus de S_j, suivant les alignements de la bibliothèque.

La fonction de *M-Coffee*

Le principe est ici toujours le même, c'est-à-dire utiliser d'autres alignements pour évaluer le problème. Mais ici, il ne s'agit plus d'utiliser des algorithmes d'alignement exacts pour évaluer l'alignement multiple. Contrairement aux deux précédentes fonctions, *M-Coffee* utilise des alignements multiples de l'ensemble des séquences de S.

En effet, *M-Coffee* 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, Dialign, ...*) pour créer 15 alignements de référence. Comme précédemment, ces résultats sont utilisés pour l'évaluation de l'alignement multiple.

La fonction de *3D-Coffee* :

Fugue [Shi *et al.*, 2001] permet de déterminer un alignement de deux séquences à partir de leurs structures tridimensionnelles.

L'évaluation dans *3D-Coffee* est obtenue à partir de la bibliothèque d'alignements de *T-Coffee*. Celle-ci est enrichie avec tous les alignements par paires obtenus en utilisant *Fugue*. Pour utiliser *3D-Coffee*, il est donc nécessaire de connaître la structure tridimensionnelle des séquences à aligner.

3.2.5 Autres fonctions

Dialign [Morgenstern, 1999] utilise des "fragments" d'alignements par paires locaux et non plus des paires de résidus. Chaque paire de fragments est évalué en fonction de sa longueur et de sa qualité. On ne peut généralement pas retrouver toutes les paires de fragments dans l'alignement multiple des séquences. La fonction objectif à maximiser est la somme des valeurs attribuées aux paires de fragments conservées dans l'alignement multiple.

La fonction *NorMD* [Thompson *et al.*, 2001] est en partie basée sur la fonction de somme des paires. Une pondération est apportée suivant la similarité des séquences mais aussi en fonction de leurs longueurs avant alignement. La valeur est normalisée afin qu'elle ne dépende que de la qualité du résultat, et non de la longueur de l'alignement ou du nombre de séquences.

3.3 Le problème d'alignement multiple de séquences

3.3.1 Définition

D'un point de vue formel, l'alignement multiple de séquences peut être représenté comme un problème de décision.

Définition 50 (Problème d'alignement multiple de séquences) *Soit S un ensemble de $k > 2$ séquences, et soit A un alignement multiple de S. Le problème d'alignement multiple de séquences consiste à déterminer si pour une fonction d'évaluation, A est le meilleur alignement multiple possible des séquences de S.*

Théorème 1 *Le problème d'alignement multiple de séquence est NP-Complet.*

La démonstration de ce théorème a été proposée en 1994 dans [Wang and Jiang, 1994]. Le problème utilisé pour la réduction polynomiale est *1-to-3-SAT*. Il s'agit d'une variante du problème *SAT* dans lequel chaque clause comporte un et un seul littéral à vrai.

3.3.2 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, la détermination de fonction des protéines.

3.3.3 Classification des algorithmes

Les algorithmes d'alignement multiples de séquences sont très nombreux. Comme nous le verrons plus loin, ils peuvent prendre des formes très différentes et être basés sur des principes très différents.

Il est toutefois possible de les regrouper selon trois classes [Notredame, 2002] en fonction de critères assez simples. Bien que le problème soit *NP-Complet*, il existe en effet quelques algorithmes exacts 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 sont alignées simultanément et les méthodes sont en revanche très différentes les unes des autres. Il est possible de décomposer à nouveau, suivant qu'il s'agit ou non d'algorithmes stochastiques.

3.4 Les algorithmes exacts

Comme nous l'avons vu précédemment, le problème d'alignement multiple de séquences est NP-Complet. Il est toutefois possible de généraliser la méthode basée sur la programmation dynamique vue au chapitre précédent. Sa forte complexité spatiale la rend inutilisable en pratique pour la plupart des problèmes d'alignement. Comme nous le montrerons plus loin, à partir de 4 séquences, la quantité de mémoire nécessaire pour réaliser les calculs

est bien souvent supérieure à ce qui est disponible dans la plupart des ordinateurs actuels. Aligner 5 séquences est considéré comme étant un petit problème, il est pourtant totalement exclu de chercher à le résoudre de façon exacte par la méthode la programmation dynamique que nous avons exposée. Nous présentons donc également ici deux algorithmes basés sur la programmation dynamique et qui utilisent chacun une heuristique. Le premier peut aligner une dizaine de séquences, et le second une vingtaine.

3.4.1 Algorithme basé sur la programmation dynamique

Dans le chapitre précédent nous avons vu comment aligner deux séquences en utilisant la programmation dynamique. Nous allons montrer maintenant que cet algorithme peut être étendu à plus de deux séquences.

Cas de 3 séquences

Le cas de 3 séquences est le plus simple pour expliquer comment réaliser un alignement multiple en utilisant le principe de la programmation dynamique. L'explication donnée ici et que nous allons principalement détailler concerne le cas le plus simple, à savoir l'alignement avec brèches à coût constant.

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 $\mathcal{P}(i, j, k)$ 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 $\mathcal{V}(i, j, k)$. Aligner S, T et U consiste donc à déterminer $\mathcal{P}(p, q, r)$ et la valeur de cet alignement est $\mathcal{V}(p, q, r)$.

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 $\mathcal{P}(p, q, r)$. La valeur de $\mathcal{V}(p, q, r)$ en fonction de ces sous-problèmes est donnée par l'équation (3.1) :

$$\mathcal{V}(p,q,r) = \max \left(\begin{array}{l} \mathcal{V}(p-1,q,r) + 2.val('s','-'), \\ \mathcal{V}(p,q-1,r) + 2.val('t','-'), \\ \mathcal{V}(p,q,r-1) + 2.val('u','-'), \\ \mathcal{V}(p-1,q-1,r) + val('s','t') + val('s','-') + val('t','-'), \\ \mathcal{V}(p-1,q,r-1) + val('s','u') + val('s','-') + val('u','-'), \\ \mathcal{V}(p,q-1,r-1) + val('t','u') + val('t','-') + val('u','-'), \\ \mathcal{V}(p-1,q-1,r-1) + val('s','t') + val('s','u') + val('t','u') \end{array} \right) \tag{3.1}$$

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 $\mathcal{V}(i,j,k)$. Il est donc nécessaire d'utiliser un cube de taille $(p+1)\times(q+1)\times(r+1)$. Les cas de base correspondent aux problèmes où une au moins des valeurs i, j ou k est égale à 0.

Le calcul des cas de base peut se ramener aux trois cas dégénérés suivants :

- $i = 0$, ce qui correspond à réaliser l'alignement des deux séquences T et U,
- $j = 0$, ce qui correspond à réaliser l'alignement des deux séquences S et U,
- $k = 0$, ce qui correspond à réaliser l'alignement des deux séquences S et T.

Ces trois alignements sont réalisés sur chacune des trois faces du cube se rejoignant au point d'origine $(0,0,0)$.

À partir des cas de base, il devient possible de remplir progressivement toute la matrice. L'algorithme se termine lorsque $\mathcal{V}(p,q,r)$ 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 Fig 3.1 montre un exemple de parcours de la matrice pour construire l'alignement.

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 progessivement remplie, jusqu'à obtenir la valeur du meilleur alignement. La construction de l'alignement correspondant s'obtient en remontant jusqu'au point origine de la matrice.

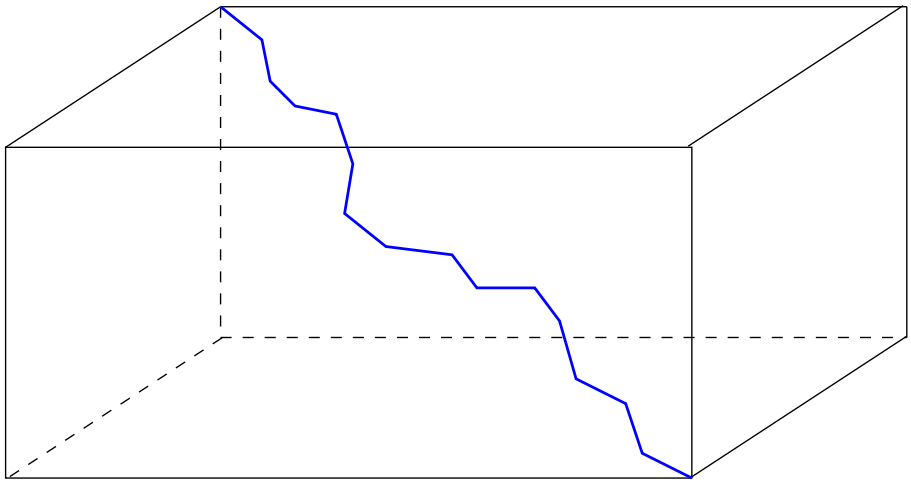

FIG. 3.1 – Construction de l'alignement pour 3 séquences.

Les limites de la programmation dynamique

Nous venons de présenter que l'algorithme d'alignement multiple de séquences basé sur la programmation dynamique est assez simple. Il est facile à mettre en œuvre, et peut être généralisé à un nombre quelconque de séquences. Pourquoi dans ce cas ne pas utiliser uniquement cet algorithme, et où est la difficulté du problème d'alignement multiple de séquences ?

Le principe de l'algorithme est assez simple, mais en revanche il est très difficilement utilisable sur les ordinateurs actuels. Nous donnons dans le tableau suivant une approximation de la mémoire nécessaire pour la construction de la matrice lors de l'utilisation du programme. 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.

Le codage des directions est ici réalisé sur un octet, ce qui permet d'avoir jusqu'à huit séquences. Les longueurs que nous avons choisies correspondent à des problèmes d'alignement de taille courante. La mémoire nécessaire pour aligner s séquences de longueur l est donc l^s octets.

Nous constatons que les ordinateurs actuels sont en mesure d'aligner jusqu'à 4 séquences si celles-ci sont de petite taille. Il sera probablement possible dans quelques années d'aligner jusqu'à 5 séquences de petite taille, mais l'on voit bien que la méthode d'alignement basée sur la programmation dynamique ne peut constituer une solution pour les problèmes plus importants en longueur ou en nombre de séquences. Comme nous allons le voir, cette méthode est tout de même utilisable si elle est couplée à des heuristiques.

	$l = 100$	$l = 200$	$l = 300$	$l = 400$	$l = 500$
$n = 4$	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 Po	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. 3.1 – *Mémoire nécessaire pour un alignement multiple.*

3.4.2 Algorithme *MSA*

Nous venons de voir que la programmation dynamique pouvait théoriquement être utilisée pour aligner un nombre quelconque de séquences. Toutefois en pratique l'algorithme ne peut pas être utilisé en raison de la mémoire qu'il requiert.

Cas de 2 séquences

L'algorithme *MSA* [Carillo and Lipman, 1988] 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. Nous montrons sur la figure 3.2 ce qui se passe en pratique ; à savoir que le chemin permettant la construction de l'alignement des deux séquences se situe à proximité de la diagonale.

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, et plus la zone centrale peut être réduite.

Sur la figure 3.2, 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 (2.1) n'est plus utilisable. Par exemple pour la coordonnée (i, j), $\mathcal{V}(i-1, j)$ n'est pas défini. Or par hypothèse, nous savons que l'alignement ne peut pas être dans la zone 2, il est donc impossible que $\mathcal{V}(i, j)$ soit obtenue à partir de cette valeur. Pour la zone limitrophe supérieure, la formule (2.1) devient donc :

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

- De même pour la zone limitrophe inférieure, la formule (2.1) devient :

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

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.

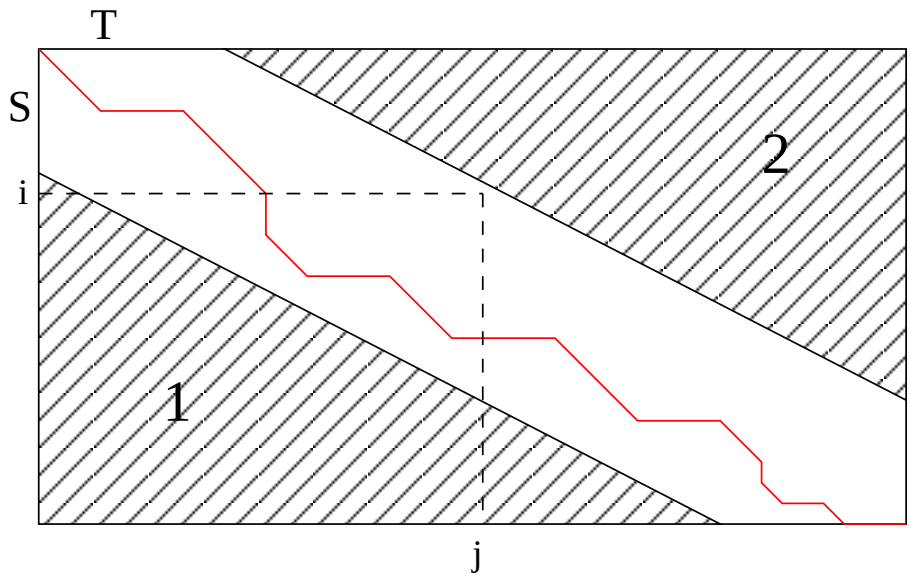

FIG. 3.2 – Restriction pour un alignement de deux séquences.

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. La représentation graphique est toutefois difficile à réaliser, même pour $n = 3$.

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. Il est ainsi possible d'aligner jusqu'à une dizaine de séquences similaires.

Remarque 6 *Cet algorithme ainsi que l'algorithme DCA que nous allons présenter sont référencés dans la littérature dans la catégorie des algorithmes exacts. Toutefois, dans les deux cas il s'agit de méthodes basées sur des heuristiques, l'appellation "algorithmes exacts" est donc abusive. L'appellation "algorithmes basés sur une méthode exacte" semble plus appropriée.*

3.4.3 Algorithme *DCA*

Introduction

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) [Stoye *et al.*, 1997].

L'intérêt d'un tel algorithme peut se comprendre facilement en regardant les valeurs données précédemment dans le tableau 3.1. Pour cela nous allons prendre un exemple simple, aligner 4 séquences de longueur 300.

D'après le tableau, la mémoire nécessaire pour réaliser l'alignement est de 8,1 Go. Or si l'on coupe chaque séquence en 3, le problème revient à aligner 3 groupes de 4 séquences de longueur 100. Ces opérations n'ayant pas à être faites simultanément, l'espace mémoire nécessaire est donc "seulement" de 100 Mo. En coupant les séquences en sous-séquences de longueur 50, l'espace mémoire nécessaire n'est plus que de 6,3 Mo.

Principe de l'algorithme

Comme nous venons de le voir il est possible de gagner un facteur 1.000 sur la quantité de mémoire en divisant les séquences en 6 sous-séquences. Et il est bien entendu possible de diviser encore la longueur de chaque séquence pour obtenir autant de problèmes de plus petite taille.

Toutefois l'algorithme *DCA* ne réalise pas uniquement un découpage des séquences, et ce pour une raison très simple. Un découpage aléatoire des séquences pose des problèmes d'alignement puisque l'on impose le début et la fin pour chacun d'eux. Le découpage ne doit donc pas être fait de cette façon.

En se basant sur les alignements par paires, l'algorithme *DCA* détermine les points de découpe de chaque séquence pour former des goupes de sous-séquences à aligner. Une fois tous les alignements de ces groupes de séquences effectués, l'algorithme les assemble pour former le résultat.

En réalisant l'alignement des goupes de sous-séquences avec l'algorithme *MSA*, il est possible de réaliser des alignements de plus de 20 séquences.

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.

3.5 Les algorithmes progressifs

3.5.1 Présentation

Comme l'indique leur nom, les algorithmes progressifs consistent à réaliser les alignements multiples en alignant progressivement les séquences. Il s'agit en quelques sortes d'algorithmes de type *divide-and-conquer*. 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. Différentes stratégies sont alors envisageables à chaque étape de ces algorithmes, même si le principe général est le même pour tous.

La première description d'un algorithme progressif a été faite dans [Hogeweg and Hesper, 1984], 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. La première étape consiste à prendre deux séquences et à les aligner. Ensuite, une par une les autres séquences sont ajoutées jusqu'à obtention du résultat. Celui-ci peut changer en fonction de l'ordre des séquences en entrée de l'algorithme.

Ce principe fondateur sera conservé, et les améliorations des algorithmes progressifs suivants porteront sur l'ordre dans lequel doivent être alignées les séquences, ainsi que la méthode pour aligner entre eux les sous-groupes de séquences déjà alignées.

La première amélioration importante est décrite dans l'algorithme de [Feng and Doolittle, 1990]. 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. *Clustal W* [Thompson *et al.*, 1994], qui reste encore actuellement [Chenna *et al.*, 2003] une référence en matière d'algorithme d'alignement pour la qualité de ses résultats ainsi que pour sa rapidité, est en très grande partie basé sur l'algorithme de *Feng et Doolittle*.

3.5.2 Profil et *Guide-Tree*

Description d'un profil

Nous l'avons vu à plusieurs reprises, l'alignement de deux séquences est un problème simple et rapide à résoudre. Il offre de plus l'avantage de posséder un algorithme exact facile à mettre en œuvre. L'utilisation d'un profil permet d'utiliser cet algorithme pour aligner simultanément deux groupes de séquences.

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. 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. À 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,

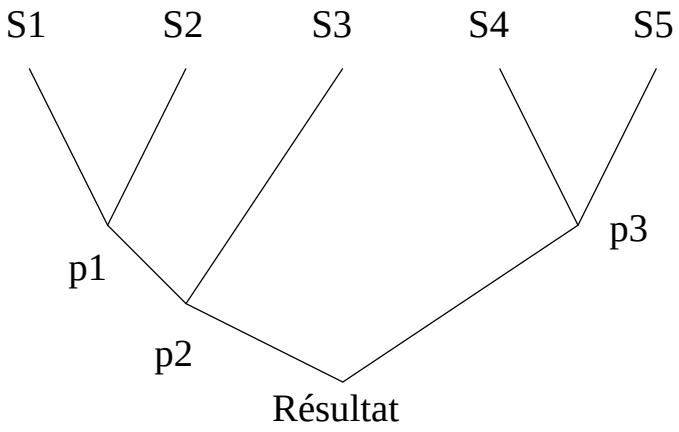

FIG. 3.3 – Exemple de *guide-tree* pour 5 séquences avec création de 3 profils p_1, p_2 et p_3.

– 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.

Utilisation d'un arbre binaire

La méthode d'alignement utilisant un profil implique qu'à partir de deux séquences, ou une séquence et un alignement, ou deux alignements il est possible d'obtenir un seul alignement par la méthode basée sur la programmation dynamique. À partir des $n > 2$ séquences à aligner, appliquer une fois cette méthode permet d'avoir $n - 2$ séquences et un alignement. En transformant cet alignement en profil, on obtient donc $n - 1$ séquences. Le processus peut ensuite être réitéré jusqu'au résultat final.

Le déroulement de cet algorithme peut donc être représenté sous la forme d'un arbre binaire. Les feuilles représentent les séquences initiales, les nœuds sont les différents profils obtenus, et la racine correspond au résultat de l'alignement multiple. L'arbre ainsi créé est appelé *guide-tree* car il permet de montrer dans quel ordre les alignements doivent être réalisés. Un exemple de *guide-tree* est donné à la figure 3.3 pour un ensemble de 5 séquences.

Création d'un *Guide-Tree*

En construisant aléatoirement un *guide-tree* pour réaliser l'alignement des séquences, cela revient à ce qui était fait dans l'algorithme de *Hogeweg*. Une solution consiste à réaliser tous les arbres binaires possibles et à construire les alignements auxquels ils conduisent. En utilisant comme critère la somme des paires (pondérée ou non), cette méthode permet de conserver le ou les meilleurs alignements. Or le nombre d'arbres binaires différents est exponentiel en fonction du nombre de séquences. Cette méthode n'est donc pas réalisable.

Il est par conséquent nécessaire 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 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 défini au chapitre II la similarité s entre deux séquences comme étant le rapport entre le nombre de correspondances d'un alignement et la longueur de l'alignement. Cette valeur vaut 1 lorsque les deux séquences sont identiques, et sera d'autant plus proche de 0 que les séquences sont différentes. Aucune correspondance dans un alignement forme donc le cas extrême où la similarité est égale à 0. La distance d entre les deux séquences est définie comme étant $d = 1 - s$. Notons, que d vérifie bien les trois conditions d'une distance au sens mathématique du terme.

En calculant les distances entre toutes les séquences, il est donc possible d'utiliser un des algorithmes permettant la création d'un arbre. Le principe global est le même pour toutes les méthodes :

1. Déterminer dans la matrice des distances les deux séquences S et T devant être reliées.
2. Créer un nœud U joignant les deux séquences S et T, de telle sorte que $d(S,U) + d(U,T) = d(S,T)$. Dans l'arbre, le nœud U représente le profil résultant de l'alignement de S et T.
3. Mettre à jour la matrice des distances en supprimant S et T, et en les remplaçant par U.
4. Si la matrice contient deux séquences ou plus, reprendre en 1.

La décroissance de la matrice des distances à chaque itération assure la terminaison de l'algorithme. À chaque étape, les calculs sont différents suivant l'algorithme utilisé : choix de U, calcul de $d(S,U)$ et $d(T,U)$, mise à jour de la matrice. Nous citons ici les deux algorithmes les plus connus :

- L'*UPGMA* [Sokal and Michener, 1958] (Unweighted Pair Group Method with Arithmetic mean) est un algorithme simple à chaque étape, puisqu'à l'étape 1 il choisit les deux séquences les plus proches, et qu'il crée ensuite U à égale distance. La distance utilisée pour cet algorithme est ultramétrique. La mise à jour de la matrice des distances est donc très simple. Il est important de noter que l'arbre obtenu ne respecte pas toujours les distances de la matrice des distances initiale.

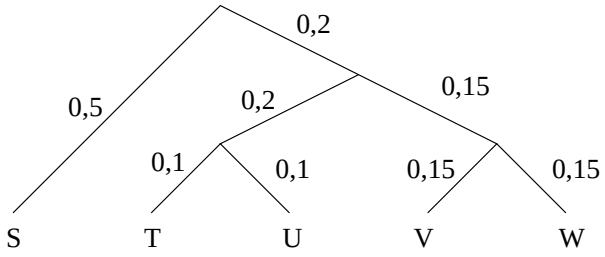

FIG. 3.4 – Exemple de *guide-tree*.

– Le *Neighbour-Joining* [Saitou and Nei, 1987] est un peu plus complexe, avec en particulier le calcul d'une divergence pour le choix des nœuds à joindre. De plus à l'étape 2 il n'y a pas nécessairement équidistance entre U et les séquences S et T. Les arbre obtenus par cette méthode sont plus proches des valeurs initiales de la matrice des distances, et donc de meilleure qualité.

La pondération

Les pondérations utilisées dans la somme des paires pondérée s'obtient à partir du *guide-tree* qui a été calculé. Ainsi pour calculer la pondération d'une séquence, il faut parcourir le chemin qui va de cette séquence jusqu'à la racine. Comme certaines parties de l'arbre vont être partagées par plusieurs chemins, il n'est pas utile de refaire plusieurs fois les mêmes calculs.

Le calcul consiste dans un premier temps à déterminer une valeur pour chaque branche de l'arbre. Ces valeurs sont utilisées pour la pondération de chaque séquence. Le principe est le suivant :

1. Déterminer pour chaque branche de l'arbre le nombre de feuilles qui y sont rattachées.
2. Diviser la longueur de la branche par le nombre obtenu à l'étape 1.
3. Pour chaque feuille parcourir le chemin conduisant à la racine, et calculer la somme des valeurs associées à chaque branche. Cette somme est la valeur attribuée à la pondération de la séquence.

Cette méthode permet d'associer à chaque séquence un poids qui dépend de sa similarité avec les autres séquences. Comme nous l'avons vu, cette pondération permet de ne pas négliger les séquences les plus distantes. En effet, lorsque des séquences sont très proches les longueurs des arcs sont faibles, et en plus divisées par le nombre de ces séquences. À l'opposé, si une séquence est très différente des autres, elle peut être reliée directement à la racine, comme sur la figure 3.4, où les valeurs représentent la longueur des branches.

Les pondérations de S et T sont les suivantes :

– $w(S) = 0,5/1 = 0,5$
– $w(T) = 0,1/1 + 0,2/2 + 0,2/4 = 0,25$

Dans cet exemple, nous constatons que la pondération associée à la séquence S est le double de la pondération de T.

3.5.3 *Clustal W*

Clustal W [Thompson *et al.*, 1994] 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.

L'alignement de deux séquences et/ou profils dans *Clustal W* fait intervenir des paramètres supplémentaires par rapport à la méthode exposée au chapitre II :

- Ainsi, pour éviter le morcellement des séquences que peuvent créer plusieurs brèches trop rapprochées un paramètre est ajouté. Il s'agit d'une distance minimale devant séparer deux brèches, par défaut cette valeur est égale à 5. Il reste bien entendu possible d'insérer deux brèches plus proches car cela peut être nécessaire, mais dans ce cas une pénalité est ajoutée au coût de la deuxième brèche.
- *Clustal W* prend également 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.

Clustal W permet d'aligner plus d'une centaine de séquences, et il donne de bons résultats dans la plupart des cas. De par sa construction, le *guide-tree* associé à n séquences nécessite de réaliser $n - 1$ alignements pour obtenir l'alignement de toutes les séquences. Le temps de calcul de *Clustal W* est donc pratiquement linéaire en fonction du nombre de séquences à aligner.

Pendant près de vingt ans, *Clustal W* a été l'algorithme de référence pour l'alignement multiple, et il reste encore aujourd'hui très utilisé.

L'algorithme général que nous avons donné peut être modifié pour donner d'autres algorithmes. Ainsi la fonction de *T-Coffee* dont nous avons parlé peut-elle être utilisée à la place de la somme des paires pondérée.

3.5.4 *MultAlin*

Comme nous l'avons déjà dit, l'ordre d'alignement des séquences est primordial pour la qualité du résultat. L'algorithme *MultAlin* [Corpet, 1988] considère que le *guide-tree*

obtenu à partir de la matrice des distances n'est pas forcément le meilleur.

Nous savons que la reconstruction de phylogénie est une des applications directes de l'alignement multiple de séquences. Comme pour les autres algorithmes progressifs, *MultAlin* commence par construire un *guide-tree* T_0 avant de déterminer un alignement multiple A_0. La différence de cet algorithme est qu'une fois le résultat obtenu, il est réutilisé pour construire un nouveau *guide-tree* T_1, plus pertinent que le premier. Ce nouvel arbre est ensuite utilisé pour construire un second alignement multiple A_1. Ce processus est réitéré jusqu'à la stabilisation, c'est-à-dire jusqu'au rang i où $T_{i-1} = T_i$.

La justification d'un tel algorithme est très simple. Le *guide-tree* initial T_0 est uniquement basé sur les alignements par paires, alors qu'à partir de T_1, les arbres obtenus prennent en compte l'ensemble des séquences.

3.5.5 *Muscle*

Présentation

L'algorithme *Muscle* [Edgar, 2004b] 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 les *k-mer* pour créer une matrice des distances. Il s'agit de toutes les sous-séquences de longueur k des alignements par paires qui s'apparient exactement. La matrice des distances est créée en dénombrant les *k-mer* de chaque paire de séquences.

Description de l'algorithme

L'algorithme de *Muscle* est divisé en trois grandes étapes, chacune de ses étapes fournissant un alignement multiple des séquences.

1. Construction du premier alignement :
   - Création d'une matrice D_1 des *distances k-mer* [Edgar, 2004a] entre toutes les paires de séquences.
   - Construction d'un *guide-tree* T_1 avec la méthode de l'UPGMA à partir de la matrice D_1.
   - Alignements par paires des séquences ou des profils pour construire un alignement multiple A_1.
2. Construction du second alignement :
   - Création d'une nouvelle matrice de distances D_2 à partir de toutes les paires de séquences de A_1.
   - Construction d'un second *guide-tree* T_2 à partir de D_2
   - Alignements par paire des séquences ou des profils pour construire un alignement multiple A_2.
3. Raffinement du résultat :
   - Choisir une branche de l'arbre.

– 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, et le *guide-tree* est mis à jour. Le point de coupure devenant la nouvelle racine.

Ces opérations sont répétées jusqu'à stabilisation de la solution ou un nombre prédéfini de fois. Les branches choisies en premier sont celles qui sont les plus proches de la racine.

Discussion

La distance *k-mer* entre deux séquences s'obtient sans avoir à réaliser un alignement. Son principal avantage est de donner une distance avec une complexité linéaire $O(n)$. Lorsque le nombre de séquences reste peu élevé la différence de temps n'est pas significative. En revanche lorsque ce nombre augmente, le temps utilisé par l'algorithme uniquement pour réaliser les alignements pas paire est très important. En effet, il y a $n-1$ alignements par paire pour créer l'alignement multiple alors qu'il faut réaliser $n^2/2$ alignements par paire pour calculer la matrice de distances.

La création du premier alignement et du deuxième *guide-tree* est au final plus rapide que la création du *guide-tree* dans *Clustal W*. De plus T_2 a été obtenu à partir d'un alignement multiple de bonne qualité, ce qui lui donne plus de sens qu'un *guide-tree* obtenu à partir d'alignements par paire.

Muscle est actuellement un des meilleurs algorithmes d'alignements multiple, tant au point de vue de la qualité des résultats que de la rapidité. En outre, il est capable d'aligner simultanément jusqu'à 5000 séquences.

3.5.6 *MAFFT*

L'algorithme *MAFFT* [Katoh *et al.*, 2002] part du même constat que *Muscle*, réaliser des alignements par paire avec une complexité en $O(n^2)$ est difficile lorsqu'il y a trop de séquences. Ce qui implique au final beaucoup de temps pour générer le premier *guide-tree*. *MAFFT* utilise une *transformée de Fourier* rapide pour construire le *guide-tree* qui lui sert pour calculer l'alignement. Cette opération permet de réduire la complexité à $O(n.ln(n))$.

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.

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

3.5.7 *ProbCons*

ProbCons [Do *et al.*, 2005] est actuellement le meilleur algorithme d'alignement multiple de séquences. Il peut être comparé à *Muscle* puisqu'il est également constitué de trois étapes similaires : construction d'un *guide-tree*, alignement et pour finir optimisation du résultat.

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

Comme dans *Muscle*, la dernière étape consiste à améliorer le résultat. Pour cela une méthode similaire a été mise en place. Il ne s'agit plus seulement de chercher à enraciner différemment le *guide-tree*. En effet, dans *ProbCons*, l'algorithme ne se contente pas de séparer les séquences en deux groupes à partir d'une branche de l'arbre. L'algorithme sépare l'alignement multiple en deux de façon aléatoire, et les réaligne. Ce processus est effectué un nombre fixé de fois. Lorsqu'un alignement de meilleure qualité est obtenu il remplace l'alignement courant.

Remarque 7 *Cette méthode ne nécessite pas de modèle particulier pour le coût des brèches. Ce coût est en effet toujours nul. Ainsi cet algorithme peut être utilisé avec moins de paramètres que les autres algorithmes progressifs, ce qui représente bien sûr un avantage important.*

3.5.8 *Dialign*

L'algorithme *Dialign* [Morgenstern, 1999] et plus récemment *Dialign-T* [Subramanian *et al.*, 2005] utilisent les *fragments*, présentés à la sous-section 3.2.5 de ce chapitre. La création de l'alignement multiple se fait en ajoutant progressivement des paires de fragments. L'algorithme procède de la façon suivante :

1. Créer la liste L de toutes les paires de fragments,
2. Evaluer toutes les paires de fragments,
3. Tant qu'il reste des paires de fragments dans L :
   (a) Prendre la paire de fragments p ayant la valeur la plus importante, et l'ajouter à l'alignement multiple.
   (b) Supprimer de L toutes les paires de fragments incompatibles avec p.

Cet algorithme ne permet généralement pas d'obtenir des résultats d'aussi bonne qualité que les algorithmes cités précédemment. Comme la création de brèches n'est pas pénalisée, l'algorithme devient compétitif lorsqu'il est nécessaire d'insérer de grandes brèches pour obtenir un alignement multiple.

3.6 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'aligne-

ment des séquences. Leur point commun est uniquement de réaliser l'alignement multiple en prenant simultanément toutes les séquences dès le début. Les stratégies utilisées sont très différentes, et nous citons principalement *SAGA*, car cet algorithme reste une référence importante.

Il est essentiel de remarquer que ces algorithmes nécessitent généralement plus de calculs, et leurs temps d'exécution sont souvent plus importants. Dans la section résultats du chapitre 5, nous donnons à titre de comparaison les temps de calculs de différents algorithmes.

3.6.1 *SAGA*

SAGA (Sequence Alignment by Genetic Algorithm) [Notredame and Higgins, 1996] est basé sur un algorithme de type génétique [Holland, 1975]. 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 schéma général suivi par l'algorithme est le suivant :

- sélection de la partie de la population devant être remplacée ; par défaut cette valeur est de 50%,
- utilisation de l'un des nombreux opérateurs les individus sélectionnés,
- mise à jour de la population en ne conservant que les cent meilleurs individus.

Ces trois opérations permettent de passer de la génération G_n à la génération G_{n+1}. L'algorithme s'arrête lorsque la population se stabilise.

Il existe 22 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. Tous les opérateurs fournissent en sortie un unique individu.

Exemple : Nous proposons ici à titre d'exemple un des opérateurs de combinaison utilisé dans l'algorithme.

Soient les deux individus suivants :

```
--SRPLIHF-GNDY          SRPLIHFGNDY------
EDRYYRENMYRY--          --E-DRYY--RENMYRY
-FGSDYEDRYY---          --F-GSDYEDRYY-----
```

faisons maintenant une "coupe droite" aléatoirement dans le premier individu :

```
--SR|PLIHF-GNDY
EDRY|YRENMYRY--
-FGS|DYEDRYY---
```

Celle-ci peut se matérialiser ainsi dans le deuxième individu :

```
SR|PLIHFGNDY------
--E-DRY|Y--RENMYRY
--F-GS|DYEDRYY----
```

En insérant des brèches, il est possible d'avoir également une coupe droite dans cet individu :

```
SR-----|PLIHFGNDY-
--E-DRY|Y--RENMYRY
--F-GS-|DYEDRYY---
```

Le nouvel individu est composé de la partie gauche du premier individu, et de la partie droite du second :

```
--SRPLIHFGNDY-
EDRYY--RENMYRY
-FGSDYEDRYY---
```

3.6.2 Autres algorithmes

Nous présentons ici trois algorithmes qui mettent en évidence la diversité des méthodes utilisées par les algorithmes itératifs.

Maximum Weigth Trace Problem

Dans [Kececioglu *et al.*, 2000], l'alignement multiple de k séquences est présenté sous la forme d'un graphe k-partite. L'ensemble V des sommets de ce graphe est formé de toutes les lettres constituant les séquences. L'ensemble E des arêtes est constitué de tous les couples de sommets appartenant à des séquences différentes. Les arêtes sont pondérées en fonctions des acides aminés qui forment leurs sommets à l'aide d'une matrice de substitution. Une *trace* du graphe est définie comme un ensemble d'arêtes représentant un alignement des séquences.

Les deux sommets d'une arête correspondent à deux lettres alignées. Pour que la trace soient un alignement, il est nécessaire que les sommets ne se croisent pas, et qu'un sommet ne soit relié au maximum qu'à un unique sommet d'une autre séquence. Ces deux conditions peuvent s'exprimer sous la forme de contraintes.

Le *Maximum Weigth Trace Problem* consiste à déterminer la trace dont la somme des pondérations est maximale. La méthode proposée exprime l'ensemble des contraintes sous la forme d'un programme linéaire en nombres entiers.

Tabu search

L'algorithme *Tabu search* [Riaz *et al.*, 2004] procède en deux étapes pour réaliser l'alignement multiple :

- création d'un alignement initial A_0 par insertion de brèches dans les séquences.
- utilisation d'une méthode tabou [Glover and Laguna, 1997] pour déplacer les brèches dans l'alignement. L'ensemble des alignements que l'on peut obtenir en déplaçant une brèche dans A_i constitue les voisins de A_i. L'alignement A_{i+1} est le meilleur voisin qui n'appartient pas à la liste *tabou*.

La solution est obtenue lorsque l'algorithme se stabilise.

Algorithme	Classe	URL
MSA	Exact	http ://searchlauncher.bcm.tmc.edu/multi-align/multi-align.html
DCA	Exact	http ://bibiserv.techfak.uni-bielefeld.de/dca/
Clustal W	Progressif	http ://bips.u-strasbg.fr/fr/Documentation/ClustalW/
MultAlin	Progressif	http ://bioinfo.genopole-toulouse.prd.fr/multalin/
Muscle	Progressif	http ://www.drive5.com/muscle/
MAFFT	Progressif	http ://align.bmr.kyushu-u.ac.jp/mafft/online/server/
ProbCons	Progressif	http ://probcons.stanford.edu/download.html
Dialign-T	Progressif	http ://dialign-t.gobics.de/
SAGA	Itératif	http ://www.tcoffee.org/Projects_home_page/saga_home_page.html
MWTP	Itératif	
Tabu Search	Itératif	http ://web.bii.a-star.edu.sg/ tariq/tabu/
SAM	Itératif	http ://www.soe.ucsc.edu/compbio/sam.html

TAB. 3.2 – *Récapitulatif des algorithmes présentés.*

Les modèles de Markov cachés

La modélisation des protéines à l'aide de modèles de Markov cachés [Haussler *et al.*, 1993] permet d'utiliser cette méthode pour l'alignement multiple de séquences. L'algorithme *SAM* (Sequence Alignment and Modeling system) [Eddy, 1995] est basé sur ce principe, les résultats ne sont toutefois pas de très bonne qualité. L'algorithme est plus efficace lorsque les séquences sont déjà alignées. Il peut alors être utilisé pour réaliser une opération d'optimisation pour un alignement multiple.

3.7 Récapitulatif des algorithmes

Nous proposons au tableau 3.2 un récapitulatif des algorithmes présentés précédemment. Pour chacun d'eux nous rappelons de quel type d'algorithme il s'agit, ainsi que l'url où il est possible de le télécharger.

3.8 Présentation des jeux d'essai

Dans cette section nous allons présenter les principales bases de jeux d'essais. Nous détaillerons plus particulièrement *Balibase* que nous avons utilisée pour valider nos algorithmes.

3.8.1 Généralités

Nous avons vu quelques exemples d'algorithmes d'alignement multiple de séquences, mais il en existe de nombreux autres. Chaque année, le nombre de publications relatives à des algorithmes d'alignement multiple avoisine la centaine. Un problème se pose alors, est-il possible de définir un algorithme donnant de meilleurs résultats, et si oui selon quel critère ?

Jusqu'à présent nous avons parlé de fonctions d'évaluation uniquement pour qu'un algorithme tente de l'optimiser au mieux. Il est néanmoins possible d'envisager avec cette méthode de comparer les résultats donnés par un ensemble d'algorithmes différents sur un ou plusieurs jeux de séquences. Il est donc simple de déterminer quel algorithme donne les meilleurs résulats.

Pourtant en pratique, il s'avère que le meilleur résultat selon une fonction d'évaluation n'est bien souvent pas le meilleur d'un point de vue biologique. Des facteurs tels que la structure tri-dimensionelle des séquences doivent être pris en compte, alors qu'ils ne sont pas accessibles lors de l'évaluation de la fonction. On constate en pratique qu'une fonction d'évaluation peut attribuer une moins bonne valeur à un alignement qui sera considéré par un biologiste comme étant le meilleur.

Les algorithmes ne sont pas destinés à donner la meilleure évaluation pour une fonction, mais à proposer un alignement qui soit le plus proche possible de ce que peut faire un biologiste. Il est donc important pour comparer les résultats obtenus de prendre 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 regroupent ces alignements, avec pour chaque jeu le meilleur résultat possible d'un point de vue biologique.

3.8.2 *Balibase*

Présentation de *Balibase*

Balibase [Thompson *et al.*, 1999] 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.

La référence 1 correspond à des jeux d'essais contenant peu de séquences. Elles sont classées par longueur (*small*, *medium*, *long*), et par pourcentage de similitude ($< 20\%$, $< 40\%$ et $> 40\%$). Les jeux d'essais de cette référence sont assez simples, et aucun algorithme ne donne de très mauvais résultats. En revanche, les autres références sont composées de jeux d'essais plus atypiques. Citons par exemple, la présence d'une séquence orpheline (Référence 2), qui n'a aucune similarité avec les autres séquences ; mais également des séquences de tailles très différentes, ou nécessitant de très grandes brèches (longueur supérieure à 100). Enfin il existe également une référence dont chacun des jeux d'essais est composé de deux ou trois sous-groupes de séquences. Chaque sous-groupe est constitué de séquences très similaires, mais par contre, les sous-groupes sont assez distants les uns des autres. Les jeux d'essais de la référence 1 sont composés de 4 à 6 séquences, alors que ceux des autres références en contiennent plus : en moyenne une dizaine de séquences, et jusqu'à 23 dans certains cas.

La version 2 de *Balibase* est basée sur la version 1. Elle reprend les 5 références existantes avec quelques modifications dans les résultats [Karplus and Hu, 2001], auxquelles sont ajoutées 4 nouvelles références. Cependant, ces nouvelles références sont uniquement

FIG. 3.5 – Principe du critère *SPS*.

constituées de jeux d'essais et ne contiennent pas de résultat optimal. Les tests que l'on trouve dans la littérature sont réalisés sur les 5 premières références de *Balibase 2*.

La version 3 de *Balibase* [Thompson *et al.*, 2005] contient également de nouveaux jeux d'essais, et quadruple ainsi le nombre total de séquences pour passer à plus de 6000. Comme pour la version 2, les nouveaux jeux ne sont pas proposés avec l'alignement optimal.

Les critères de comparaison

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*, pour *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. 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. À noter que les brèches ne sont pas comptabilisées, il est donc possible qu'un alignement multiple ait un score de 1 sans être identique à la référence.

La fonction de comparaison *CS*, pour *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 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. 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.

Il est facile de constater que le critère de comparaison *CS* est beaucoup plus contrai-

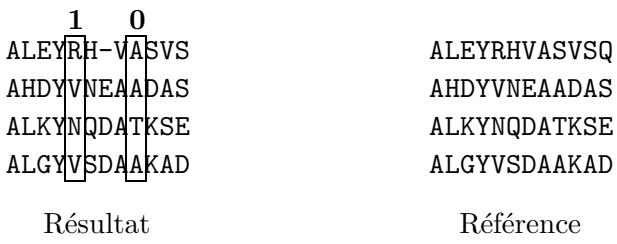

FIG. 3.6 – Principe du critère *CS*.

gnant que le critère *SPS*. En effet, il suffit d'un seul résidu mal placé pour que le reste de la colonne soit évalué à 0. Ce phénomène est particulièrement visible pour les jeux d'essais avec une séquence orpheline. Cette séquence étant difficile à aligner avec les autres la valeur *CS* de l'alignement peut très facilement valoir 0.

Le site Internet de *Balibase* (`http ://bips.u-strasbg.fr/fr/Products/Databases /Balibase`) propose un programme écrit en langage C (`bali_score.c`) qui permet d'évaluer les valeurs *SPS* et *CS*.

Les résultats

Pour chacun des deux critères de comparaison que nous venons de voir, il y a deux façons de réaliser l'évaluation des résultats. La première est celle que nous avons vue jusqu'à maintenant, c'est-à-dire en considérant l'alignement multiple dans son intégralité. La deuxième évaluation concerne uniquement une ou plusieurs parties de l'alignement. En effet, dans un alignement, seules certaines parties sont intéressantes d'un point de vue biologique.

Les descriptions de ces parties intéressantes de l'alignement sont appelées des annotations. Un fichier d'annotation est un fichier texte contenant des informations sur un alignement multiple, et en particulier la position de début et la position de fin de chacune de ces parties. D'autres informations sont également contenues, comme par exemple la répartition des séquences en familles.

Les deux méthodes sont en fait importantes, car si un algorithme ne donne de bons résultats que sur une petite partie de l'alignement, cela peut provenir d'un manque de robustesse. Inversement, l'objectif des algorithmes est de pouvoir être utilisés par des biologistes. Il faut donc que les résultats sur les parties annotées soient bons. Les valeurs qui sont données dans les publications sont celles obtenues en prenant en compte le fichier d'annotation. La fonction `bali_score.c` peut prendre en paramètre optionnel un fichier d'annotation pour le calcul des critères *SPS* et *CS*.

Remarque 8 *Il est important de remarquer que le site de Balibase contient également un tableau dans lequel sont déjà calculés des résultats. Ainsi pour 10 algorithmes différents les résultats sont proposés pour chaque jeu d'essais. Ces résultats sont destinés à simplifier les tests sur un nouvel algorithme, et permettent de le comparer aux 10 algorithmes en question. Toutefois, aucune conditon expérimentale n'est précisée, et il est impossible de*

reproduire les résultats indiqués. Cette idée de proposer les résultats est intéressante, mais elle ne peut pas être utilisée dans la pratique.

3.8.3 Les autres bases

Nous présentons ici deux autres bases très connues. Comme nous le verrons dans les chapitres suivants, nous avons réalisé nos tests avec les jeux d'essais de *Balibase*. Pour cette raison nous donnons ici moins de détails sur *Homstrad* et *Oxbench*.

Homstrad

La première version de *Homstrad* [Mizuguchi *et al.*, 1998] est antérieure à *Balibase*, mais les jeux d'essais proposés étaient moins diversifiés. En effet, ils correspondent uniquement à la Référence 1 de *Balibase*.

Actuellement la version existante de *Homstrad* [Stebbings and Mizuguchi, 2004] est beaucoup plus importante. Elle contient plus d'un millier d'alignements multiples, chacun étant dédié à une famille de protéines. Tous les alignements ont été obtenus à partir de séquences dont la structure tridimensionnelle est connue.

À noter toutefois que plus de la moitié des jeux d'essais ne sont composés que de 2 ou 3 séquences, et une très grande partie en contiennent moins de 8.

Oxbench, Prefab, SABmark

Plusieurs autres bases d'alignements multiples plus récentes existent, nous les citons sans détailler car elles sont souvent construites sur le même principe.

- *Oxbench* [Raghava *et al.*, 2003]
- *Prefab* [Edgar, 2004b]
- *SABmark* [Walle *et al.*, 2005]

3.9 Conclusion

Dans le chapitre précédent nous avons étudié le problème d'alignement de deux séquences. Nous venons de montrer que ce problème était un cas particulier et qu'il pouvait être généralisé à un nombre quelconque de séquences. Aligner un ensemble S de $k > 2$ séquences est une transformation qui consiste à ajouter des brèches dans les séquences de S. Le problème d'alignement multiple de séquences consiste donc à déterminer le meilleur alignement possible.

Pour cela, il existe des fonctions d'évaluation qui permettent d'associer une valeur qualitative à un alignement. Les fonctions existantes sont des modèles mathématiques, et aucune ne permet de modéliser exactement la réalité biologique.

Le problème de l'alignement multiple de séquences est très important en biologie. Or il s'agit d'un problème *NP-Complet*, et bien que la méthode basée sur la programmation dynamique puisse être généralisée, elle ne peut pas être utilisée en pratique. Pour cette

raison de nombreux algorithmes ont été développés, et ils peuvent être classés 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 alignent quant à eux toutes les séquences simultanément.

Des bases de jeux d'essais ont été créées pour pouvoir comparer les différents algorithmes existants. Elles permettent également de tester les résultats de tout nouvel algorithme et de le comparer à ceux existants. Les jeux sont constitués de plusieurs séquences, et à chaque fois le meilleur alignement est donné. À l'aide de critères de comparaison, il est ainsi possible de classer les algorithmes en fonction des leurs résultats.

Le temps mis pour aligner les séquences est également un critère important. Du point de vue du biologiste, le temps nécessaire pour obtenir un alignement est bien souvent déterminant pour l'utilisation d'un algorithme. Même si la qualité du résultat est prépondérante, pour des algorithmes permettant d'obtenir des résultats de qualités équivalentes, le plus rapide est considéré comme étant le meilleur.

Deuxième partie

Chapitre 4

Plasma I : un algorithme d'alignement multiple par insertion de blocs

4.1 Présentation générale

Dans la première partie nous avons présenté le problème de l'alignement multiple de séquences ainsi que plusieurs algorithmes. Les algorithmes les plus compétitifs actuellement ne permettent pas de déterminer l'alignement optimal d'un point de vue biologique. Nous proposons donc dans cette partie deux nouveaux algorithmes d'alignement multiple de séquences.

Dans ce chapitre, nous présentons *Plasma* (*Progressive Local Search for Multiple Alignment*) [Derrien *et al.*, 2002; Derrien *et al.*, 2003b; Derrien *et al.*, 2003a] un algorithme d'alignement progressif. Comme pour les méthodes progressives traditionnelles, notre algorithme va réaliser un alignement progressif de l'ensemble des séquences. En effet, à chaque itération de l'algorithme, uniquement un sous-ensemble de séquences est aligné. Toutefois l'alignement de deux ensembles de séquences préalablement alignées est effectué au moyen d'une méthode globale. Celle-ci prend en effet l'intégralité du sous-ensemble de séquences pour réaliser l'alignement.

L'algorithme peut être décomposé en deux étapes principales. Dans un premier temps, l'algorithme réalise tous les alignements de deux séquences afin de créer une matrice des distances. Cette matrice permet de déduire, sous la forme d'un guide-tree, l'ordre dans lequel les séquences sont alignées au cours de la seconde partie de l'algorithme. Dans cette étape nous utilisons une méthode de recherche locale pour aligner deux sous-ensembles de séquences. L'algorithme général est le suivant :

1. Alignement par paires avec la méthode de programmation dynamique :
   (a) Alignement global de chaque couple de séquences,
   (b) Création d'une matrice des distances,
   (c) Calcul du *guide-tree* qui en résulte.

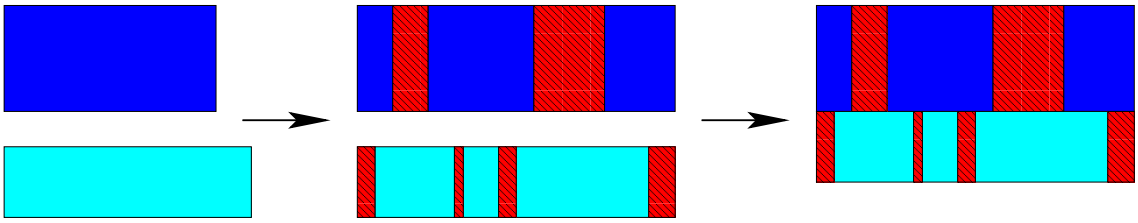

FIG. 4.1 – Principe de l'algorithme.

2. Alignement itératif et progressif au moyen d'une méthode de recherche locale :
   (a) Choisir les deux groupes de séquences les plus proches,
   (b) Les aligner en utilisant une méthode de recherche locale,
   (c) Les fusionner pour former un nouveau groupe ; mettre à jour la matrice des distances avec ce groupe,
   (d) Arrêter lorsque toutes les séquences sont alignées, sinon reprendre en 2.a.

La première étape de l'algorithme est une analyse des séquences qui est commune aux méthodes d'alignement globales. Pour un ensemble de k séquences à aligner, l'algorithme calcule les $k.(k-1)/2$ alignements par paires optimaux en utilisant la méthode basée sur le principe de la programmation dynamique. L'évaluation de chacun de ces alignements permet d'obtenir une matrice de distances. Le *guide-tree* qui permettra de déterminer l'ordre d'alignement des groupes de séquences est calculé à partir de la matrice des distances. La méthode utilisée ici pour construire l'arbre est celle de l'*UPGMA*.

La seconde étape correspond à la partie principale de l'algorithme. Il s'agit ici de construire de façon itérative l'alignement complet, par alignements successifs de deux groupes de séquences préalablement alignées. Par extension, une séquence seule est également considérée comme étant un groupe. En suivant l'ordre d'alignement indiqué par le *guide-tree*, l'algorithme prend les deux groupes de séquences les plus proches, pour former un nouveau groupe de séquences alignées. L'alignement est réalisé au moyen d'une méthode de descente, en insérant ou en supprimant des brèches dans l'un des deux alignements. Le groupe de séquences alignées ainsi obtenu peut dès lors être à son tour aligné avec un autre groupe. L'algorithme se termine lorsque les k séquences sont alignées dans le même groupe.

Cette méthode offre l'avantage de permettre de conserver toutes les séquences lors de l'alignement, sans avoir recours à la notion de profil. L'évaluation de l'alignement se fait donc en conservant chaque séquence dans sa totalité.

L'alignement de deux groupes est fait à partir d'une méthode de recherche locale. Le couple constitué par les deux groupes de séquences à aligner forme la configuration initiale. Ce couple constitue un premier alignement qui forme une solution acceptable pour notre algorithme. Lorsque les deux groupes de séquences ne sont pas de la même longueur, une brèche est ajoutée dans le plus court. L'algorithme transforme ensuite la configuration initiale par insertions ou suppressions de brèches dans un des deux groupes, ou dans les deux simultanément. Lorsqu'il n'est plus possible d'améliorer cette configuration, toutes les séquences sont regroupées pour former la solution.

4.2 La recherche locale

4.2.1 Configuration et espace de recherche

Soient G_1 et G_2 deux groupes de séquences, et soient A_1 et A_2 deux alignements de G_1 et G_2 respectivement. Nous dirons que le couple (A'_1, A'_2) est une configuration si, et seulement s'il vérifie les conditions suivantes :

1. L'union de A'_1 et A'_2 forme un alignement multiple des séquences de $G_1 \cup G_2$,
2. La suppression des colonnes complètes de brèches présentent dans A'_1 et A'_2 permet de retrouver A_1 et A_2.

L'espace de recherche pour les deux groupes de séquences G_1 et G_2 est composé de tous les alignements (A'_1, A'_2) vérifiant ces deux conditions. Nous utilisons une méthode de recherche pour déterminer localement le meilleur alignement possible.

La configuration initiale c_0 est obtenue directement à partir de A_1 et de A_2. Si A_1 et A_2 sont de même longueur, alors $c_0 = (A_1, A_2)$. Si les alignements ne sont pas de la même taille, des colonnes de gaps sont insérées à la fin du plus court des deux afin que l'union vérifie les critères d'un alignement multiple de séquences.

Nous appellerons longueur d'une configuration la longueur des séquences de l'alignement multiple correspondant.

4.2.2 La fonction d'évaluation

La fonction d'évaluation que nous utilisons pour déterminer la qualité d'une configuration est la fonction de somme des paires. Pour déterminer le coût de chaque paire de résidus, nous utilisons la matrice *Pam* 250. En effet, celle-ci offre un bon compromis lorsque l'on ne possède pas d'indication sur la nature des jeux de séquences.

Le calcul du coût des brèches est effectué suivant le modèle de coût affine $K + l.h$, avec $K = -10$ et $h = -1$.

Nous pouvons définir la fonction d'évaluation utilisée comme la somme des évaluations de chaque paire de séquences. Aucun coefficient de pondération n'est pris en compte, toutes les séquences sont donc considérées ici comme ayant la même importance.

4.2.3 Description d'un voisinage

Soit $c = (A_1, A_2)$ une configuration du problème. Le voisinage $N(c)$ de c est défini en ajoutant une colonne de brèches complète dans la configuration. Soit g la largeur de cette colonne, il existe trois possibilités pour réaliser cette insertion :

- Insertion simple d'une brèche dans A_1,
- Insertion simple d'une brèche dans A_2,
- Insertion simultanée d'une brèche dans A_1 et d'une brèche de même largeur dans A_2 à des positions différentes.

Il y a deux possibilités pour faire l'insertion simple dans A_1. En effet, l'insertion d'une brèche dans A_1 nécessite l'ajout simultané d'une brèche dans A_2. Il y a pour cela deux

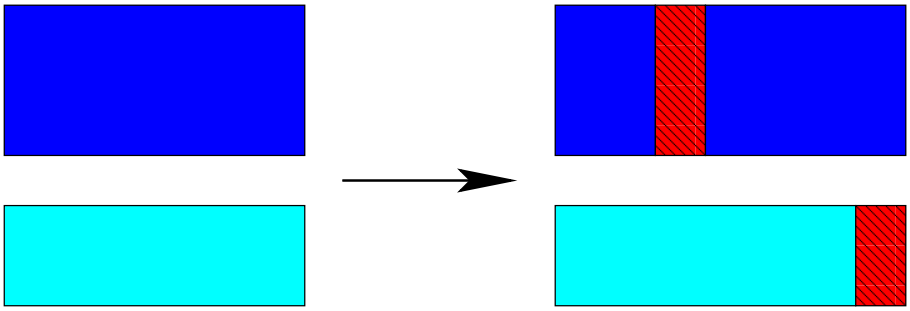

FIG. 4.2 – Première insertion d'une brèche dans A_1.

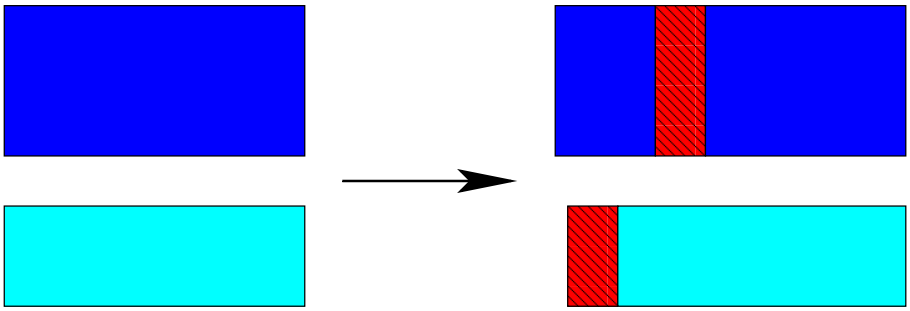

FIG. 4.3 – Seconde insertion d'une brèche dans A_1.

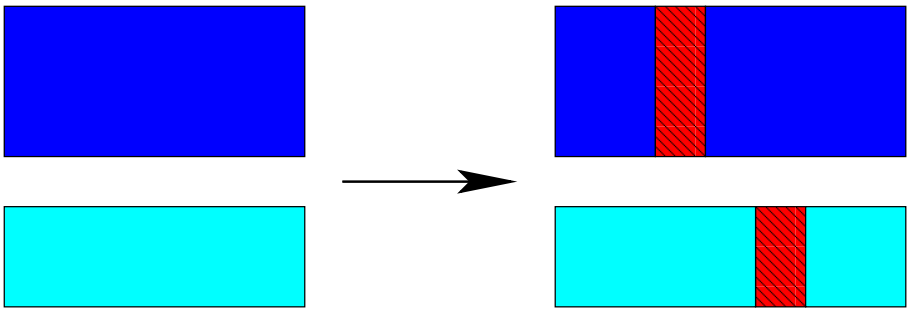

FIG. 4.4 – Insertion d'une brèche dans A_1 et dans A_2.

possibilités, suivant que cette brèche est ajoutée au début (FIG. 4.2) ou à la fin de A_2 (FIG. 4.3).

La double insertion dans A_1 et dans A_2 se fait comme indiqué à la figure FIG. 4.4.

L'insertion d'une colonne de brèches doit permettre de transformer c en une nouvelle configuration acceptable. Il faut pour cela que l'union des deux alignements après insertion donne toujours un alignement. Les deux conditions qui doivent être vérifiées pour conserver un alignement sont les suivantes :

- Si une colonne de brèches n'a été insérée que dans un seul alignement il faut également ajouter une colonne dans l'autre alignement. Cette nouvelle insertion est faite soit au début, soit à la fin. Le choix retenu est celui qui optimise la fonction d'évaluation.
- Après une insertion, il faut supprimer toutes les colonnes de brèches communes aux deux groupes.

Remarque 9 *Le deuxième point permet lorsque l'on effectue une insertion simultanée dans A_1 et dans A_2, de supprimer en partie ou entièrement une brèche insérée lors d'une itération précédente (*FIG. 4.5*). Cette méthode permet de raffiner la qualité de l'alignement, en effet les premières brèches insérées permettent de positionner sommairement les différentes zones contenant des similarités. Au fur et à mesure que l'on réalise simultanément des insertions dans les deux alignements, les brèches initiales sont mieux réparties pour positionner les zones de plus faible similarité.*

Le nombre de voisins d'une configuration c dépend directement de l'intervalle de valeurs dans lequel varie la largeur g de la colonne de brèches insérée. L'algorithme contient un paramètre g_{max}, qui correspond à la valeur maximale que peut prendre g. La valeur de g est donc un entier compris dans l'intervalle $[1, g_{max}]$. Le nombre de voisins pour une configuration est donné en fonction de g_{max} et de la longueur n de la configuration. Le voisinage d'une configuration peut être subdivisé selon le type d'insertion considéré. L'insertion d'une colonne de brèches dans A_1 nécessite d'insérer une colonne de brèches de même largeur au début ou à la fin de A_2. Réciproquement, nous avons bien sûr les mêmes insertions pour A_2 et A_1. Pour chacune des trois possibilités d'insertion, nous avons :

- $2.n.g_{max}$ insertions d'une colonne de brèches dans A_1,
- $2.n.g_{max}$ insertions d'une colonne de brèches dans A_2,

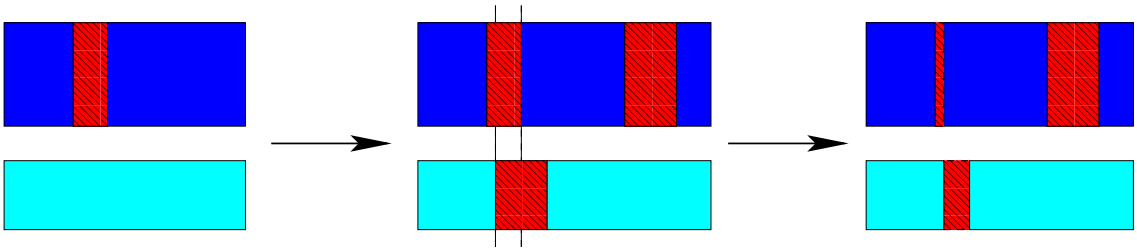

FIG. 4.5 – Principe de suppression d'une partie de brèche par double insertion.

– $n.(n+1).g_{max}$ insertions simultanées dans A_1 et dans A_2.

La valeur attribuée par défaut à g_{max} a été déterminée de façon empirique. Une valeur comprise entre 5 et 10 permet d'obtenir de bons résultats, tout en limitant l'espace de recherche. Pour des valeurs inférieures à 5, l'espace de recherche semble trop restreint et la qualité des résultats obtenus est moins bonne. L'utilisation de valeurs supérieures à 10 pour g_{max} semble n'avoir que peu d'effet sur le résultat final. Toutefois il peut sembler utile de choisir cette valeur en fonction des séquences à aligner. Si les alignements par paires de la première étape montrent une forte similarité entre les séquences, il n'est pas utile d'avoir un g_{max} élevé. Si en revanche il faut aligner des séquences assez distantes il est préférable de pouvoir augmenter la taille maximale que peuvent avoir les brèches.

4.2.4 La méthode de descente

Soit $c = (A_1, A_2)$ une configuration, et soit $N(c)$ l'ensemble des voisins de c. Notre algorithme utilise la fonction de somme des paires pour évaluer chaque configuration de $N(c)$. L'algorithme va donc essayer de faire toutes les insertions possibles de brèches ayant une taille comprise entre 1 et g_{max}, soit dans A_1, soit dans A_2, soit dans A_1 et A_2 simultanément.

Soit c' le meilleur des voisins de c au sens de la fonction de somme des paires. Si c' est meilleur que c, il devient la configuration courante et remplace c. La recherche de l'optimum peut alors recommencer à partir de cette nouvelle configuration. Lorsqu'il n'est plus possible de trouver un voisin c' meilleur que c, la partie descente s'arrête. A_1 et A_2 sont alors fusionnés pour créer un nouvel alignement.

4.3 Complexité de l'algorithme

Le nombre d'itérations de l'algorithme dépend entièrement du jeu de séquences à aligner, et il n'est pas possible de le prévoir avant. Il n'est par conséquent pas possible de donner une complexité globale à l'algorithme. Nous pouvons toutefois donner la complexité au niveau de la partie recherche locale pour le passage d'une configuration c à la configuration suivante c'.

Il faut prendre en premier lieu la complexité de la fonction d'évaluation, ce qui permet de donner la complexité pour déterminer la valeur d'un voisin de c. Soient k_1 et k_2 le

nombre de séquences de A_1 et A_2 respectivement.

Chaque résidu de A_1 doit être comparé avec les $n.k_2$ résidus de A_2. Au total cela fait donc $n^2.k_1.k_2$ comparaisons pour évaluer une configuration de $N(c)$. Le nombre de configurations est :

- $2.n.g_{max}$ pour les insertions dans A_1,
- $2.n.g_{max}$ pour les insertions dans A_1,
- $n.(n+1).g_{max}$ pour les insertions dans A_1 et A_2 simultanément,

La complexité pour l'évaluation d'une configuration est donc en $O(n^4.k_1.k_2.g_{max})$. Cette complexité théorique de notre algorithme pourrait être diminuée en ne recalculant pas à chaque fois l'intégralité de la somme des paires. Il faudrait pour cela calculer localement la différence de coût occasionnée par le déplacement d'une position pour la brèche.

4.4 Résultats expérimentaux

4.4.1 Les paramètres du programme

Comme indiqué précédemment, l'algorithme dépend de quatre paramètres. Le premier g_{max} fixe la taille maximale que peut avoir une brèche lors d'une insertion. Si l'alignement nécessite l'insertion d'une brèche de taille supérieure, plusieurs brèches peuvent être insérées successivement.

Taille maximale des insertions

g_{max} est le seul paramètre propre à l'algorithme. L'idéal est de pouvoir choisir une valeur très grande, de façon à ce qu'une brèche de taille quelconque puisse être insérée. Toutefois le nombre de voisins d'une configuration donnée, dépend directement de la valeur de ce paramètre. Nous venons de voir avec la complexité que le nombre d'insertions possibles et donc le nombre de voisins est de l'ordre de $g_{max}.n^2$. Pour cette raison, nous avons choisi une valeur de g_{max} égale à 10 pour tester l'algorithme.

Les autres paramètres

Nous avons choisi les valeurs -10 et -1 pour K et h, et la matrice de substitution que nous avons prise est la matrice *Pam* 250.

4.4.2 Les jeux d'essai

Description des jeux d'essais

Nous avons réalisé des essais sur 5 groupes de séquences de tailles différentes. Ces jeux d'essais sont des exemples communs puisqu'ils sont cités dans certains articles ou fournis comme exemples avec des programmes existants. Tous les exemples sont composés de séquences de protéines. Les principales caractéristiques sont données dans le tableau 4.1.

	Nombre de séquences	Longueur (acides aminés)	Nature des séquences
Exemple 1	5	153	hémoglobine
Exemple 2	6	164	lysozyme
Exemple 3	14	187	aspartic proteinase
Exemple 4	15	264	prions
Exemple 5	17	153	hémoglobine

TAB. 4.1 – Jeux d'essais utilisés.

Remarques sur les jeux de tests :

- Les séquences contenues dans les jeux d'essai de prions sont très similaires. Elles forment de plus un piège pour les logiciels d'alignement puisque certaines zones sont répétées plusieurs fois. Différentes variantes sont citées dans des articles, avec des nombres de séquences différents.
- L'exemple 1 est constitué d'un sous-ensemble de séquences de l'exemple 5. Nous avons choisi de le conserver car il présente une caractéristique intéressante dont nous parlerons dans la partie des résultats.
- Les exemples 2, 3 et 5 sont trois des jeux d'essais de SAGA (voir §3.5.1).

L'implémentation de cet algorithme résulte d'un travail préliminaire [Derrien *et al.*, 2002] et a été faite en deux programmes indépendants. Il était nécessaire d'effectuer une transformation manuelle du résultat du premier programme avant de pouvoir utiliser le second, nous n'avons par conséquent que peu de tests.

4.4.3 Résultats

Comparaison avec d'autres programmes

Les programmes avec lesquels nous allons comparer nos résultats sont principalement ceux qui étaient les plus performants à l'époque de l'algorithme : *Clustal W* et *SAGA*.

Les comparaisons sont faites en prenant pour tous les résultats la même fonction objectif : la somme des paires. Celle-ci est calculée avec la matrice de coût *PAM 250* et les valeurs $K = -10$ et $h = -1$ pour l'évaluation du coût des brèches. Il s'agit donc ici d'un problème de maximisation.

	Clustal W	*SAGA*	*Plasma*
Exemple 1	2928	2951	**2986**
Exemple 2	2997	2869	**3493**
Exemple 3	3556	860	**6164**
Exemple 4	132033	131171	**132259**
Exemple 5	19234	19498	**23228**

TAB. 4.2 – Évaluation des alignements avec somme des paires.

Commentaires sur les résultats

Le tableau 4.2 présente les résultats obtenus, rappelons qu'une valeur élevée correspond à un résultat de bonne qualité. Sur les cinq jeux d'essais que nous avons utilisés, les évaluations avec la somme des paires donnent à chaque fois le meilleur résultat pour notre programme. Les écarts sont parfois très importants puisque pour l'exemple 3, notre programme donne une valeur 7 fois plus importante par rapport au résultat de *SAGA*. Dans cet exemple, *Clustal W* qui donne souvent de très bons résultats est lui aussi assez distancé avec un pourcentage de différence de 73,3%.

Dans l'exemple 4 où les séquences sont très similaires, on constate que les résultats sont eux aussi très proches.

La différence de résultats peut bien évidemment être due à la fonction objectif. En effet, nos comparaisons sont faites avec la fonction de somme des paires, qui est la fonction utilisée par notre algorithme. Cette fonction n'est pas celle qu'utilisent les deux autres algorithmes (la somme des paires pondérée) pour réaliser leurs alignements. Toutefois nous pouvons faire quelques remarques sur les résultats :

- Dans l'exemple 1, nous avons constaté que les valeurs des dix alignements par paires de *Clustal W* étaient inférieures ou égales aux valeurs des alignements par paires de notre programme. En multipliant les résultats des deux algorithmes par des coefficients de pondérations qui sont positifs, les résultats de notre algorithme restent donc supérieurs. En faisant la somme, c'est-à-dire en calculant la somme des paires pondérées, notre algorithme donne donc un meilleur résultat que *Clustal W* sur cet exemple.
- Dans l'exemple 4, les séquences étant très proches la pondération ne doit intervenir que très peu. L'évaluation avec l'une ou l'autre des fonctions objectif ne doit donc pas changer beaucoup le résultat.
- Les temps de calcul de notre programme sont voisins de ceux de *SAGA* pour les exemples traités. Les exemples 1 et 2 demandent moins de 3 minutes, alors que les trois autres exemples nécessitent entre une heure et une heure et demie.

4.5 Conclusion

Dans ce chapitre nous avons présenté *Plasma* un algorithme d'alignement multiple de séquences. Cet algorithme présente des similitudes avec les autres algorithmes progressifs, comme par exemple la construction du guide-tree. Toutefois nous constatons une différence importante, car chaque étape de l'alignement est réalisée en prenant en compte toutes les séquences, comme le fait un algorithme itératif.

Le cœur proprement dit de l'algorithme permet de réaliser l'alignement de deux groupes de séquences alignées. Le principe est simple puisqu'il s'agit à chaque itération d'insérer au mieux une brèche dans un des deux groupes de séquences ou dans les deux simultanément. Pour cela, toutes les configurations possibles d'insertions sont évaluées. L'insertion qui est réalisée est celle qui augmente le plus possible la valeur de la fonction objectif.

Pour cela l'algorithme suit une méthode de descente. Les deux groupes de séquences initiaux forment la configuration initiale. Le voisinage de cette solution est réalisé en

effectuant toutes les insertions possibles d'une colonne de brèches dans les deux groupes. La nouvelle configuration est celle qui améliore le plus le résultat, et l'algorithme se termine lorsqu'il n'y a plus de voisin en mesure d'améliorer le score.

Une nouvelle implémentation de cet algorithme est nécessaire afin de pouvoir le valider sur les jeux d'essais de *Balibase*. Il sera alors possible de le comparer aux autres algorithmes avec les critères *SPS* et *CS*. Il serait également intéressant d'envisager une autre méthode que la descente pour réaliser l'alignement, par exemple un recuit simulé [Aart and van Laarhoven, 1987] ou une méthode tabou.

Chapitre 5

Plasma II : un algorithme d'alignement multiple de séquences sans profil

5.1 Introduction

Dans Clustal W les séquences alignées sont remplacées par un profil. L'utilisation de ce profil masque les séquences réelles, risquant par là même une perte de l'information et donc une perte de la qualité. Avec *Plasma II* [Derrien *et al.*, 2005; Richer *et al.*, 2007] nous proposons un algorithme de type progressif basé sur le principe de la programmation dynamique.

L'utilisation d'un profil pour réaliser l'alignement multiple de séquences offre l'avantage d'être une méthode simple et rapide à utiliser. En effet, l'algorithme utilisé pour aligner simultanément une séquence et un profil ou deux profils est le même que celui utilisé pour l'alignement de deux séquences. La complexité de l'alignement de deux ensembles de séquences alignées avec les profils est donc la même que celle de l'algorithme d'alignement par paire : $O(n.m)$ où n et m sont les longueurs respectives des deux ensembles de séquences à aligner. Les résultats obtenus sont de bonne qualité lorsque les séquences sont assez similaires. Toutefois, le compromis qui doit être fait dans le cas d'une faible similarité engendre le risque de produire des profils de mauvaise qualité. Le profil n'est pas totalement représentatif des séquences à partir desquelles il est formé puisque l'information qu'il doit retranscrire est trop générale. Pour cette raison l'alignement qui en découle risque de perdre en qualité. Des erreurs d'alignement dues au manque de précision des profils vont se répercuter, la qualité des profils obtenus diminuant ainsi à chaque alignement. Plus le nombre de séquences à aligner est important, et plus l'alignement final perdra en qualité.

5.2 Principe de l'algorithme

L'algorithme développé dans *Plasma II* a été réalisé en partant de ce constat. Comme l'alignement risque de perdre en qualité à cause des compromis réalisés lors de la création

des profils, nous avons décidé de ne pas utiliser cette méthode. Les alignements successifs réalisés par l'algorithme sont effectués en conservant à chaque itération toutes les séquences.

De plus, il s'agit d'un problème proposé il y a quelques années [Kececioglu and Zhang, 1998] et n'ayant pas eu de réponses satisfaisantes. L'utilisation de profils vise d'une part à diminuer la complexité de l'algorithme et d'autre part il semble que l'expression de l'algorithme de programmation dynamique sans simplification par profil soit très difficile à résoudre.

Plasma II est basé sur le même principe général que *Plasma* I. La grande différence réside dans la méthode utilisée pour aligner les deux blocs. *Plasma* I utilise une méthode de descente pour réaliser des insertions de colonnes de brèches. En revanche *Plasma II* est un algorithme basé sur la programmation dynamique.

Comme pour les autres algorithmes progressifs, *Plasma II* commence par réaliser un *guide-tree* en utilisant la méthode du Neighbour-Joining (cf. page 66). Le premier niveau d'alignement dans l'arbre correspond aux nœuds dont les fils sont associés aux séquences initiales. La méthode utilisée pour créer ces nœuds est celle du *Neighbour-Joining*. Les résultats obtenus par cet algorithme ne sont en revanche pas transformés en profil. Nous conservons l'intégralité des séquences de chacun des alignements sous forme de "blocs". Le terme de bloc correspond à l'alignement d'un sous-ensemble de séquences. Le bloc constitue une structure pour l'alignement, de sorte qu'il puisse être réaligné par la suite. Lorsqu'un bloc est aligné avec une séquence ou un autre bloc, toutes les insertions de brèches se font entre les colonnes du bloc. Un bloc est un alignement qui peut à son tour être aligné avec d'autres séquences, mais dont le contenu de chaque colonne reste inchangé. L'algorithme de *Plasma II* détermine comment insérer des brèches à l'intérieur d'un bloc pour que l'alignement soit optimal.

La raison principale de l'utilisation de cette méthode vient de l'ordre d'alignement des séquences. Comme celui-ci se fait suivant une décroissance de la similarité entre les blocs, chaque paire de résidus préalablement alignés doit être conservée. Nous supposons donc ici que toute colonne de bloc précédemment obtenue doit être conservée. L'algorithme ne permet pas de remettre en cause un résultat obtenu lors de l'obtention d'un bloc. Ce raisonnement ne prend donc pas en compte les similarités qui peuvent exister localement entre des séquences distantes.

5.3 Extension de la programmation dynamique dans *Plasma II*

Afin de réaliser l'alignement de deux blocs ou d'un bloc avec une séquence, nous avons redéfini les opérations d'édition conventionnelles (section 2.3). La justification n'est bien entendu pas la même, puisque pour deux séquences cela permet d'expliquer l'évolution de l'une à l'autre. Or ici il n'est pas possible de considérer ces opérations de la sorte puisque les deux blocs ne contiennent pas nécessairement le même nombre de séquences.

En effet, nos opérations d'édition étendues nous servent uniquement à définir une méthode de résolution. Pour deux séquences nous avons vu que ces opérations sont utilisées

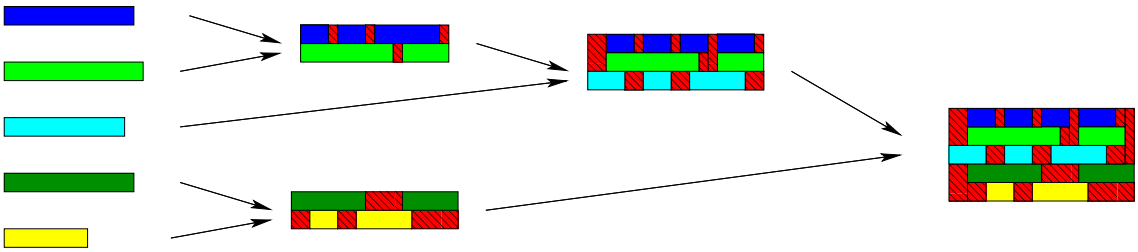

FIG. 5.1 – Exemple d'alignement par blocs pour un jeu de 5 séquences. Le résultat est obtenu par une construction progressive de sous-alignements.

pour définir la valeur associée aux différents couples de résidus possibles. La généralisation des opérations d'édition aux blocs nécessite de définir des opérations spécifiques non plus pour 2 résidus mais pour deux colonnes de résidus. Là encore, il faut pouvoir définir des valeurs pour chaque type d'opération. La somme des valeurs obtenues pour chacune des opérations d'édition étendues associées à l'alignement des deux blocs doit être égale à la valeur de l'alignement multiple de ces deux blocs. Ainsi en optimisant le résultat à chaque pas de l'alignement des deux blocs, il est possible d'optimiser la valeur globale du résultat.

Soit B un bloc de longueur l composé de n séquences définies sur un alphabet Σ. Nous noterons B_i la colonne i de ce bloc, et B_{ij} le contenu de la ligne j de B_i. Soit $\Sigma' = \Sigma \cup \{-\}$ et w la fonction définie sur $\Sigma' \times \Sigma'$ qui à deux éléments de Σ' associe leur évaluation.

Soit $B(p)$ l'évaluation de la colonne B_p.

$$B(p) = \sum_{i=1}^{i=n-1} \sum_{j=i+1}^{j=n} w(B_{pi}, B_{pj})$$

Les opérations *match* et *mismatch* étendues aux blocs consistent à mettre en regard une colonne de chaque bloc. L'évaluation de ces opérations se fait en ajoutant les évaluations de chacune des deux colonnes avec la valeur de l'évaluation d'une colonne par rapport à l'autre. Chacune de ces trois évaluations est obtenue avec les opérations d'édition classiques.

Soient B et B' deux blocs de longueurs l et l' composés respectivement de n et n' séquences. La valeur associée à l'opération d'édition *match* ou *mismatch* étendue pour l'alignement des blocs est donnée par :

$$M(B_p, B'_q) = B(p) + B'(q) + \sum_{i=1}^{i=n} \sum_{j=1}^{j=n'} w(B_{pi}, B'_{qj})$$

Les opérations *insertion* et *deletion* étendues aux blocs consistent à mettre en regard une colonne d'un bloc avec une colonne de brèche. Dans le cas de l'opération d'insertion, la brèche est mise dans le premier bloc, et la valeur associée est donnée par :

$$I(B'_q) = B'(q) + n. \sum_{j=1}^{j=n'} w(-, B'_{qj})$$

De façon symétrique, on obtient pour l'opération de deletion :

$$D(B_p) = B(p) + n'.\sum_{i=1}^{i=n} w(B_{pi}, -)$$

Exemple : Aligner les deux blocs suivants avec des brèches à coût constant.

```
AGCCT
ATCC-
A-CGT
```

```
AGCTAG
--CTAC
```

Pour cela nous utilisons la matrice suivante :

$\mathcal{M}$	$-$	A	C	G	T
$-$	/	1	1	1	1
A	1	6	3	3	3
C	1	3	6	3	3
G	1	3	3	6	3
T	1	3	3	3	6

Les calculs pour les problèmes de plus bas niveau ainsi que la deuxième ligne donne les résultats suivants :

			-	A	G	C	T	A	G
			-	-	-	C	T	A	C
-	-	-	0	4	8	20	32	44	53
A	A	A	24	40	44	56	68	80	89
G	T	-	33	**49**					
C	C	C	57						
C	C	G	75						
T	-	T	87						

Le calcul $\mathcal{V}(2,1)$ se fait de la façon suivante :

$$\begin{pmatrix} - \\ - \\ - \\ A \\ - \end{pmatrix} = 4, \quad \begin{pmatrix} G \\ T \\ - \\ - \\ - \end{pmatrix} = 9, \quad \begin{pmatrix} G \\ T \\ - \\ A \\ - \end{pmatrix} = 15$$

Ce qui permet d'obtenir le résultat suivant :

$$\mathcal{V}(2,1) = \max \begin{pmatrix} \mathcal{V}(2,0) + 4, \\ \mathcal{V}(1,1) + 9, \\ \mathcal{V}(1,0) + 15 \end{pmatrix} = \begin{pmatrix} 33 + 4, \\ 40 + 9, \\ 24 + 15 \end{pmatrix} = 49$$

Définition : Soient B_1 et B_2 deux blocs de séquences sur un alphabet Σ ayant pour longueurs respectives m et n. Soient $P(i,j)$ le problème consistant à aligner les deux sous-blocs $B_1[1..i]$ et $B_2[1..j]$, $i \leq m$ et $j \leq n$, et $D(i,j)$ la distance d'édition étendue pour les blocs associés. Le problème de l'alignement multiple par blocs peut être défini de la façon suivante :

- le problème consiste à déterminer $P(m,n)$,
- le sous-problème $P(i,j)$ consiste à calculer $D(i,j)$,
- l'initialisation se fait avec $P(i,0)$ et $P(0,j)$.

La construction récursive de la matrice D au moyen de cette extension de la méthode de programmation dynamique, assure qu'à chaque étape, selon la fonction de somme des paires pondérée, la valeur est optimale. En effet, les cas de base $P(i,0)$ et $P(0,j)$ correspondent à l'alignement d'un élément de Σ avec $-$. Ensuite le calcul du problème $P(i,j)$ est effectué à partir des sous-problèmes $P(i,j')$, $j' < j$ et $P(i',j)$, $i' < i$. Or par hypothèse de récursivité, ces sous-problèmes ont une solution optimale.

Les brèches sont évaluées en utilisant la méthode affine classique. Le coût d'ouverture de la brèche est noté K et son coût d'extension h.

5.4 Algorithme général

L'algorithme général de *Plasma II* se déroule en plusieurs étapes distinctes. Nous proposons donc ici uniquement une version simplifiée de l'algorithme :

Fonction aligner(E : liste de séquences)
Début
    Construire un *guide-tree* de E par la méthode du *Neighbour-Joining*
    Construire n blocs composés chacun d'une séquence
    $L = \{B_1, \ldots B_n\}$
    Tant que $\mid L \mid \neq 1$ **faire**
        Choisir les 2 blocs B_i et B_j les plus proches d'après le *guide-tree*
        $B =$ aligner(B_i, B_j)
        $L = L \cup \{B\} / \{B_i, B_j\}$
    Fin
    Return B
Fin

5.5 Détails de l'implémentation

5.5.1 Présentation générale de l'algorithme

L'algorithme de *Plasma II* utilise la fonction de somme des paires pondérée (section 3.1) pour évaluer la qualité d'un alignement multiple. L'évaluation des brèches au niveau de l'alignement multiple se fait en utilisant une fonction affine. L'algorithme que nous proposons est donc une généralisation de l'algorithme existant pour deux séquences vu au chapitre II, mais utilisant les opérations d'éditions étendues que nous venons de présenter. Pour faire le parallèle avec l'algorithme utilisant un coût de brèche constant, nous évoquons également ici la possibilité de l'étendre pour réaliser l'alignement de deux blocs de séquences. Pour aligner deux séquences, cet algorithme donne des résultats de moins bonne qualité que l'algorithme avec brèches à coût affine, nous n'avons donc pas essayé de l'implémenter.

5.5.2 Principe d'alignement de deux blocs

Alignement de deux blocs à coût de brèche constant

Le principe de cet algorithme est le même que celui pour l'alignement de deux séquences avec coût de brèche constant. Il construit progressivement une matrice de valeur, en se basant à chaque fois sur les trois sous-problèmes directs. L'objectif ici n'est pas de maximiser la valeur de l'alignement de deux séquences, mais de maximiser la valeur de la somme des paires pour l'alignement de deux blocs. La valeur à chaque étape est calculée au moyen des opérations d'édition étendues.

À chaque étape de la construction de la matrice de valeurs, l'algorithme doit chercher la meilleure opération possible entre :

- La mise en regard de deux colonnes issues des deux blocs,
- L'insertion d'une colonne de brèches dans le premier bloc,
- L'insertion d'une colonne de brèches dans le deuxième bloc.

Ensuite, la construction de l'alignement se fait suivant le même principe que celui que nous avons vu pour deux séquences.

Alignement de deux blocs à coût de brèche affine

Il s'agit ici d'étendre l'algorithme d'alignement de deux séquences avec brèches à coût affine à l'alignement de blocs. Il n'est toutefois pas possible d'adapter directement le principe de cet algorithme car les séquences utilisées ne contiennent initialement aucune brèche. Or ce n'est plus le cas lorsqu'il s'agit de l'alignement de blocs. Par définition, un bloc est le résultat de l'alignement d'au moins deux séquences, et il contient des brèches dans la plupart des cas. Une création de brèche dans le cas de deux séquences doit toujours avoir un coût de K. En revanche pour l'alignement par blocs, une création de brèche peut avoir un coût différent selon la séquence. Si la position précédente dans la séquence n'est pas une brèche, l'algorithme est dans le cas normal de création d'une brèche, avec un coût

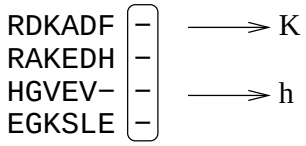

FIG. 5.2 – Insertion d'une colonne de brèche dans un bloc.

de K. Si par contre la position précédente est une brèche, le coût pour cette séquence doit être celui d'une extension, à savoir h.

Ainsi, dans l'exemple de la figure 5.2, toutes les brèches de la colonne insérée ont un coût de K, sauf la troisième qui a un coût de h.

L'algorithme utilisé reprend le principe que nous avons présenté pour l'alignement de deux blocs avec coût de brèche constant. L'objectif est ici encore d'évaluer la mise en regard d'une colonne de chaque bloc, ainsi que l'insertion et la deletion par ajout d'une colonne de brèches dans l'un puis dans l'autre des blocs.

Pour ne pas avoir à rechercher à chaque fois le contenu des colonnes précédentes, lors d'un calcul, il est possible de créer deux tableaux pour chaque bloc de séquences contenant des pré-calculs. Le premier T_{norm} contient la valeur calculée de chaque colonne, en prenant en compte toutes les brèches. Ce tableau peut être calculé progressivement à partir du début du bloc. Ensuite, chaque brèche est évaluée une fois en fonction de la colonne précédente. Le second tableau T_{gap} effectue également le calcul complet de la valeur du bloc, mais en supposant à chaque colonne qu'une colonne de brèche complète a été insérée. Une fois ces deux tableaux calculés pour chacun des deux blocs, il ne reste plus qu'à faire les calculs entre les deux blocs. Il convient donc de prendre en compte à chaque fois l'opération d'édition utilisée précédemment. Selon les cas il faut utiliser l'un ou l'autre des deux tableaux.

Le reste du calcul concerne l'évaluation entre les deux blocs. Il faut différencier les cas, suivant qu'il y a insertion ou deletion ou qu'il y a une opération de mise en regard de deux colonnes. Chacun des cas nécessite de connaître l'opération précédente :

- Pour une opération d'insertion, plusieurs cas sont possibles :
  - lorsque la position de la colonne du premier bloc contient un caractère, l'insertion a un coût de K s'il s'agit d'une ouverture de brèche dans le deuxième bloc, et h sinon.
  - lorsque la position de la colonne du premier bloc contient une brèche, l'insertion a un coût de 0. Si cette configuration est utilisée comme sous-problème pour chacune des colonnes suivantes des deux blocs, ces deux brèches ne doivent pas être prises en compte pour une extension de brèche.

  Les 3 cas possibles sont présentés à la figure 5.3.
- l'opération de deletion est similaire à l'opération d'insertion, en inversant le rôle joué par chacun des deux blocs. Les évaluations possibles sont les mêmes.
- Pour l'opération de match/mismatch, plusieurs cas sont également possibles :
  - Lorsque les deux positions pour lesquelles il faut calculer la valeur ne sont des

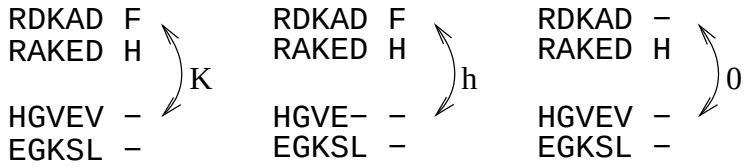

FIG. 5.3 – L'évaluation d'une colonne de brèches insérée dans le deuxième bloc dépend de la présence ou non d'autres brèches dans le premier bloc.

brèches dans aucun des deux blocs, la valeur est obtenue directement dans la matrice de substitution $\mathcal{M}$.

- Si une seule des deux positions est une brèche, il faut déterminer s'il s'agit d'une ouverture ou d'une création de brèche. Le coût est de h si une colonne de brèche a été insérée précédemment ou si la colonne contient une brèche. Dans le cas contraire le coût associé à cette position sera de K.
- Si les deux positions sont des brèches, alors le coût associé est nul. Si l'on ne considère que les deux séquences en question, l'alignement par paire qu'elles constituent contient une double brèche. Il convient donc de l'ignorer, et celle-ci ne doit pas intervenir pour la position suivante.

La mise en équations du problème doit prendre en compte tous les facteurs exposés ci-dessus par rapport aux équations obtenues précédemment pour le cas de l'alignement de deux séquences avec coût de brèches affine. Soient $\mathcal{B}_1$ et $\mathcal{B}_2$ deux blocs à aligner. Nous appellerons $\mathcal{B}(\alpha)$ la séquence α du bloc $\mathcal{B}$ et $\mathcal{B}(\alpha, i)$ le $i^{\text{ème}}$ caractère de la séquence α du bloc (B). Enfin, appelons T^1_{norm} et T^1_{gap} les tableaux définis précédemment associés à $\mathcal{B}_1$ et, T^2_{norm} et T^2_{gap} ceux associés à $\mathcal{B}_2$.

Avec ces notations nous pouvons redéfinir les équations présentées dans la section 2.4.1.

Calcul de $DD(i, j)$: Le calcul de la valeur de $DD(i, j)$ est basé sur l'équation obtenue pour deux séquences, elle prend en compte les remarques qui viennent d'être données pour être adaptée à deux blocs.

$$DD(i, j) = DM(i - 1, j - 1) + T^1_{norm}(i) + T^2_{norm}(j) + \mathcal{V}(i/j)$$

où $\mathcal{V}(i/j)$ correspond à l'évaluation de la $i^{\text{ème}}$ colonne de $\mathcal{B}_1$ avec la $j^{\text{ème}}$ colonne de $\mathcal{B}_2$. Cette valeur est définie par :

$$\mathcal{V}(i/j) = \sum_{\alpha \in \mathcal{B}_1} \sum_{\beta \in \mathcal{B}_2} v(\mathcal{B}_1(\alpha, i), \mathcal{B}_2(\beta, j))$$

où la fonction v est définie par :

- $\mathcal{M}(\mathcal{B}_1(\alpha, i), mathcal B_2(\beta, j))$ si $\mathcal{B}_1(\alpha, i)$ et $\mathcal{B}_2(\beta, j)$ ne sont pas des brèches,
- K si soit $\mathcal{B}_1(\alpha, i)$ soit $\mathcal{B}_2(\beta, j)$ est une ouverture de brèche,
- h si soit $\mathcal{B}_1(\alpha, i)$ soit $\mathcal{B}_2(\beta, j)$ est une extension de brèche,
- 0 si $\mathcal{B}_1(\alpha, i)$ et $\mathcal{B}_2(\beta, j)$ sont des brèches.

Calcul de $DH(i,j)$: Le calcul de $DH(i,j)$ pour l'alignement de deux blocs est également basé sur l'équation obtenue pour deux séquences. En utilisant les notations définies précédemment on obtient l'équation suivante :

$$DH(i,j) = HM(i-1,j-1) + T^1_{norm}(i) + T^2_{norm}(j) + \mathcal{V}(i/j)$$

Calcul de $DV(i,j)$: De façon similaire, il est possible d'étendre le calcul de $DV(i,j)$ à deux blocs :

$$DV(i,j) = VM(i-1,j-1) + T^1_{norm}(i) + T^2_{norm}(j) + \mathcal{V}(i/j)$$

Calcul de $DM(i,j)$: Comme pour l'alignement de deux séquences, la valeur de $DM(i,j)$ est définie comme étant le maximum des valeurs de la famille $\mathcal{D}$.

$$DM(i,j) = \max\{DD(i,j), DH(i,j), DV(i,j)\}$$

Calcul de $HD(i,j)$: La valeur de $HD(i,j)$ correspond au coût d'insertion d'une colonne complète de brèches dans le bloc $\mathcal{B}_1$. Cette insertion est faite à la suite d'une opération d'édition de type match/mismatch. Le coût de la colonne j de $\mathcal{B}_2$ est donc celui indiqué dans le tableau T^2_{norm}. La colonne de brèche a quant à elle un coût nul.

$$HD(i,j) = DM(i,j-1) + T^2_{norm}(j) + \mathcal{V}'(i/j)$$

où $\mathcal{V}'(i/j)$ correspond à l'évaluation d'une colonne de brèches insérée après la $i-1^{\text{ème}}$ colonne de $\mathcal{B}_1$ avec la $j^{\text{ème}}$ colonne de $\mathcal{B}_2$. Cette valeur est définie par :

$$\mathcal{V}'(i/j) = \sum_{\alpha \in \mathcal{B}_1} \sum_{\beta \in \mathcal{B}_2} v'(\mathcal{B}_1(\alpha,i), \mathcal{B}_2(\beta,j))$$

Comme pour le calcul dans la famille de matrices $\mathcal{D}$, on définit la fonction v' suivant les différents cas possibles :

- K si $\mathcal{B}_2(\beta,j)$ est un caractère et que l'insertion dans le bloc $\mathcal{B}_1$ est une ouverture de brèche, c'est-à-dire que $\mathcal{B}_1(\alpha,i-1)$ est un caractère.
- h si $\mathcal{B}_2(\beta,j)$ est un caractère et que l'insertion dans le bloc $\mathcal{B}_1$ est une extension de brèche, c'est-à-dire que $\mathcal{B}_1(\alpha,i-1)$ est également une brèche.
- 0 si $\mathcal{B}_2(\beta,j)$ est une brèche.

Calcul de $HH(i,j)$: La valeur de $HH(i,j)$ correspond au coût d'une extension d'une colonne de brèches dans le bloc $\mathcal{B}_1$. Donc aucune colonne de brèches n'est insérée entre les colonnes $j-1$ et j de $\mathcal{B}_2$. La valeur de la $j^{\text{ème}}$ colonne de $\mathcal{B}_2$ est donc $T^2_{norm}(j)$. La valeur de $HH(i,j)$ est donc donnée par :

$$HH(i,j) = HM(i,j-1) + T^2_{norm}(j) + \mathcal{V}''(i/j)$$

où $\mathcal{V}''(i/j)$ correspond à l'évaluation de l'extension de la colonne de brèches insérée après la $i-1^{\text{ème}}$ colonne de $\mathcal{B}_1$ avec la $j^{\text{ème}}$ colonne de $\mathcal{B}_2$. Cette valeur est définie par :

$$\mathcal{V}''(i/j) = \sum_{\alpha \in \mathcal{B}_1} \sum_{\beta \in \mathcal{B}_2} v''(\mathcal{B}_1(\alpha, i), \mathcal{B}_2(\beta, j))$$

où v'' est la fonction définie par :
- h si $\mathcal{B}_2(\beta, j)$ est un caractère,
- 0 sinon.

Calcul de $HV(i,j)$: La valeur de $HV(i,j)$ correspond au coût d'une insertion de brèche entre les colonnes $i-1$ et i du bloc $\mathcal{B}_1$, alors qu'une colonne de brèches a été insérée précédemment entre la $j-1^{\text{ème}}$ et la $j^{\text{ème}}$ colonnes de $\mathcal{B}_2$. Ce cas est identique à celui du calcul de $HD(i,j)$ dans le cas de 2 séquences, et il en est de même pour l'alignement de 2 blocs :

$$HV(i,j) = DM(i,j-1) + T^2_{gap}(j) + \mathcal{V}'(i/j)$$

En effet, la fonction v' définie pour $HD(i,j)$ peut de nouveau être utilisée. La différence avec le calcul de $HD(i,j)$ correspond à la présence de brèches entre les colonnes $j-1$ et j. Or l'incidence que cela peut avoir se fait uniquement s'il faut évaluer des brèches dans la colonne j de $\mathcal{B}_2$. Mais comme on met en regard une colonne de brèches, dans tous les cas le coût associé est nul.

Calcul de $HM(i,j)$: Comme pour l'alignement de deux séquences, la valeur de $HM(i,j)$ est définie comme étant le maximum des valeurs de la famille $\mathcal{H}$.

$$HM(i,j) = \max\{HD(i,j), HH(i,j), HV(i,j)\}$$

Calcul de $VD(i,j)$: Le calcul de $VD(i,j)$ est symétrique au calcul de $HD(i,j)$. En effet, l'insertion de la colonne de brèches est également faite après une opération de type match/mismatch, mais cette fois-ci dans le bloc $\mathcal{B}_2$. Dans le bloc $\mathcal{B}_1$, la valeur de la colonne i est obtenue dans le tableau T^1_{norm}. La fonction V' peut être utilisée en inversant le rôle de i et celui de j.

$$VD(i,j) = DM(i-1,j) + T^1_{norm}(i) + \mathcal{V}'(j/i)$$

Calcul de $VH(i,j)$: Le calcul de $VH(i,j)$ correspond à une insertion de brèches dans le bloc $\mathcal{B}_2$ suite à une insertion dans $\mathcal{B}_1$. De même que $HD(i,j)$ et $HV(i,j)$ sont similaires, nous avons également une similarité entre $VH(i,j)$ et $VD(i,j)$. Ainsi, on obtient le résultat suivant :

$$VH(i,j) = HM(i-1,j) + T^1_{gap}(i) + \mathcal{V}'(j/i)$$

Calcul de $VV(i,j)$: Le calcul de $VV(i,j)$ correspond à une extension de brèche dans le bloc $\mathcal{B}_2$. Ce cas est donc similaire au cas présenté pour le calcul de $HH(i,j)$. La valeur de $VV(i,j)$ est donnée par :

$$VV(i,j) = VM(i-1,j) + T^1_{norm}(i) + \mathcal{W}''(j/i)$$

où $\mathcal{W}''(i/j)$ correspond à l'évaluation de l'extension de la colonne de brèches insérée après la $j-1^{\text{ème}}$ colonne de $\mathcal{B}_2$ avec la $i^{\text{ème}}$ colonne de $\mathcal{B}_1$. Cette valeur est définie par :

$$\mathcal{W}''(i/j) = \sum_{\alpha \in \mathcal{B}_1} \sum_{\beta \in \mathcal{B}_2} w''(\mathcal{B}_1(\alpha,i), \mathcal{B}_2(\beta,j))$$

Où w'' est la fonction définie par :
- h si $\mathcal{B}_1(\alpha,i))$ est un caractère,
- 0 sinon.

Calcul de $VM(i,j)$: La valeur de $VM(i,j)$ est définie comme étant le maximum des valeurs de la famille $\mathcal{V}$.

$$VM(i,j) = \max\{VD(i,j), VH(i,j), VV(i,j)\}$$

Calcul de $M(i,j)$: La valeur de $M(i,j)$ est définie comme le maximum de toutes les matrices, c'est-à-dire :

$$M(i,j) = \max\{DM(i,j), HM(i,j), VM(i,j)\}$$

5.5.3 La pondération

L'ensemble des équations que nous venons d'établir permet de définir complètement chaque pas de calcul de l'algorithme de *Plasma II*. Toutefois, pour ne pas surcharger les équations présentées, nous avons choisi de ne pas faire intervenir les pondérations. Pour obtenir une somme des paires pondérées, il est nécessaire de les faire intervenir dans certaines parties des équations obtenues, et il suffit pour cela de multiplier toutes les valeurs par les pondérations des deux séquences qui ont permis de les obtenir. Par exemple pour la séquence α du bloc $\mathcal{B}_1$ et la séquence β du bloc $\mathcal{B}_2$, il convient de remplacer dans les équations les valeurs suivantes :
- $\mathcal{M}(S_\alpha(i), T_\beta(j))$ devient $\mathcal{M}(S_\alpha(i), T_\beta(j)) \times w(S_\alpha) \times w(T_\beta)$,
- K devient $K \times w(S_\alpha) \times w(T_\beta)$,
- h devient $h \times w(S_\alpha) \times w(T_\beta)$,

où w est la fonction qui permet d'associer un poids à une séquence.

Dans la version que nous avons implémentée de *Plasma II*, le poids d'une séquence se calcule comme dans Clustal W à partir du *guide-tree* de l'alignement.

5.6 Complexité de l'algorithme :

Nous proposons ici le calcul de la complexité pour la construction des tableaux T_{norm} et T_{gap} ainsi que pour chaque valeur des différentes matrices. Toutefois, pour les matrices DM, HM, VM et M la complexité concerne le calcul d'une recherche de maximum de 3 valeurs. Il s'agit donc d'une complexité constante qui sera négligée par rapport aux autres calculs.

5.6.1 Complexité pour les calculs de T_{norm} et T_{gap}

Soient n_1 et n_2 les nombres de séquences composant respectivement $\mathcal{B}_1$ et $\mathcal{B}_2$. Pour un bloc $\mathcal{B}$ de longueur l et composé de n séquences, nous avons :

- $n.(n-1)/2$ calculs pour chaque colonne d'un tableau,
- l colonnes à calculer pour chaque tableau.

Au total pour tous les tableaux, nous avons $n_1.(n_1-1).l_1 + n_2.(n_2-1).l_2$ calculs, soit une complexité en $O(n_1^2.l_1 + n_2^2.l_2)$.

5.6.2 Complexité pour le calcul des matrices

Exemple : complexité du calcul de $DD(i,j)$

Cherchons la complexité pour la construction des matrices avec comme exemple le calcul de $DD(i,j)$. Nous avons vu que $DD(i,j)$ était défini par :

$$DD(i,j) = DM(i-1,j-1) + T^1_{norm}(i) + T^2_{norm}(j) + \mathcal{V}(i/j)$$

- Les trois premières valeurs de cette égalité s'obtiennent en temps constant,
- $\mathcal{V}(i/j)$ est défini comme l'évaluation du premier bloc par rapport au second. On obtient n_2 calculs pour chaque élément de la ligne i de $\mathcal{B}_1$. Soit au total une complexité en $O(n_1.n_2)$.

La complexité du calcul de $DD(i,j)$ est donc en $O(n_1.n_2)$. Soit au final pour le calcul complet de la matrice DD une complexité de $O(n_1.n_2.l_1.l_2)$.

Cas général : complexité pour les autres matrices

Evaluons la complexité pour les autres matrices, suivant le type d'opération d'édition auquel elles correspondent :

- Le calcul des matrices HD et VD est similaire à celui de DD car il s'agit également d'opérations de type match/mismatch.
- En raison de la présence d'une colonne de brèches, les matrices DH, DV, HH, HV, VH et VV nécessitent moins de calculs que les matrices de type match/mismatch. La complexité reste toutefois en $O(n_1.n_2.l_1.l_2)$.
- Les valeurs $DM(i,j)$, $HM(i,j)$, $VM(i,j)$ et $M(i,j)$ sont obtenues en temps constant, donc la complexité pour le calcul complet de ces matrices est en $O(l_1.l_2)$.

La complexité générale pour l'alignement de deux blocs de longueurs l_1 et l_2 composés respectivement de n_1 et n_2 séquences est donc de la forme :

$$O(n_1^2.l_1 + n_2^2.l_2 + n_1.n_2.l_1.l_2 + l_1.l_2)$$

Or nous avons $l_1.l_2 = o(n_1.n_2.l_1.l_2)$, et dans le cas général $n_1 \ll l_2$ et $n_2 \ll l_1$. On obtient donc une complexité de la forme :

$$O(n_1.n_2.l_1.l_2)$$

5.6.3 Complexité globale de l'algorithme

L'algorithme global est composé de deux principales parties, la création de la matrice des distances et le calcul de tous les alignements de blocs. Cherchons la complexité globale de l'algorithme pour un alignement de $n > 2$ séquences. Soit l la longueur de la séquence la plus longue.

Complexité de la matrice des distances :

La complexité pour l'alignement de deux séquences est en $O(l^2)$. Pour calculer les $n(n-1)/2$ alignements, la complexité devient donc $O(n^2.l^2)$.

Pour la construction du *guide-tree*, nous avons besoin de rechercher à chaque nœud la valeur minimale de la matrice des distances. Ce qui conduit à une complexité en $O(n^3)$. Comme il ne prend pas en compte la longueur des séquences, lorsque $n \ll l$, le calcul de l'arbre est négligeable.

Complexité des alignements de blocs :

Nous allons chercher la complexité dans le pire des cas pour l'alignement successif de toutes les séquences.

Montrons que le maximum de $n_1.n_2$ avec $n = n_1 + n_2$, $n_1 \geq 0$ et $n_2 \geq 0$ est obtenu pour $n_1 = n/2$. Nous avons en effet :

- $n_1.n_2 = n_1.(n - n_1)$
- $n_1.n_2 = n.n_1 - n_1^2$

En dérivant par rapport à n_1 cette expression, nous obtenons $n - 2.n_1$. Le maximum de $n_1.n_2$ est donc obtenu pour $n - 2.n_1 = 0$, c'est-à-dire pour $n_1 = n/2$.

La complexité du pire des cas correspond donc à l'alignement de l'ensemble des séquences, réparties de façon égale dans les deux blocs. La complexité de cette opération est donc en $O(n^2.l^2/4)$, c'est-à-dire $O(n^2.l^2)$.

L'alignement de n séquences en utilisant la méthode d'alignement par blocs nécessite $n - 1$ alignements successifs. On peut en effet montrer cela par récurrence, puisque la propriété est vraie pour $n = 2$. Et si elle est vraie pour n, en ajoutant une séquence, il y a un alignement de plus à réaliser.

La complexité de la partie correspondant à l'alignement des blocs est donc de la forme $O(n^3.l^2)$.

Complexité de l'algorithme de *Plasma II* :

La complexité de la partie d'alignements par paire pour réaliser la matrice des distances est négligeable devant la complexité des alignements par bloc. De même la complexité de la construction du *guide-tree* est très inférieure. La complexité de *Plasma II* est donc donnée par :

$$O(n^3.l^2)$$

5.7 Alignement d'alignements et NP-Complétude

Nous devons à Rumen Andonov d'avoir suscité notre attention quant au fait que [Ma *et al.*, 2003; Kececioglu and Zhang, 1998] ont démontré que le problème qui consiste à aligner deux alignements en utilisant la somme des paires ainsi qu'une fonction de score affine pour les gaps était NP-Complet.

Cela est contradictoire avec le résultat que nous obtenons qui indiquerait que le problème est polynomial.

Il n'existe pas de démonstration formelle dans [Kececioglu and Zhang, 1998] et la démonstration de [Ma *et al.*, 2003] est bien trop sommaire et sybiline pour être comprise. Cependant, Kececioglu prétend disposer d'un logiciel appelé *alignalign* qui aligne de manière optimale deux alignements. Ce logiciel [Wheeler and Kececioglu, 2007] est d'une complexité en $O(5^k.n^2)$ dans le pire des cas, mais l'auteur note qu'en pratique, la complexité observée est en $O(k^2.n^2)$.

Des expériences que nous avons menées sur des jeux d'essais générés de manière aléatoire ont montré que *alignalign* n'est capable de trouver l'alignement optimal que dans 5 % des cas. *alignalign* est n'a jamais été en mesure de trouver de score meilleur que les scores que nous avons pu obtenir.

D'après notre compréhension, il semble que la démonstration de la NP-Complétude du problème provienne de la manière de compter les brèches. Les démonstrations de [Kececioglu and Zhang, 1998] et [Ma *et al.*, 2003] reposent sur une évaluation basée sur la comptabilité *quasi naturelle* (quasi natural gap count) introduite par Altschul [Altschul, 1989b]. Pour notre part, nous évaluons les brèches à la manière de Gotoh qui est directe. Les auteurs observent que l'utilisation de l'évaluation quasi naturelle ne permet pas de déterminer de manière simple le début d'une brèche, on est alors contraint de parcourir la séquence pour déterminer le début de la brèche ce qui implique plus de calculs que nécessaire. Dans le cas ou les séquences ne contiennent que peu de brèches, l'ensemble de ces calculs "*inutiles*" est diminué et on observe alors une complexité polynomiale.

5.8 Résultats expérimentaux

Nous présentons dans cette section les jeux d'essai sur lesquels nous avons testé l'implémentation de *Plasma II*.

5.8.1 Présentation des jeux d'essai

Nous avons testé notre implémentation de *Plasma II* sur les catégories 1 à 5 de *Balibase 2*. Ces jeux d'essais servent de références et sont communément utilisés par les autres algorithmes d'alignement multiple. Il est donc possible de comparer directement les résultats obtenus par *Plasma II* par rapport aux résultats des autres algorithmes.

Les résultats sont présentés pour chaque famille ou sous-famille de problèmes, sous la forme d'une moyenne des valeurs obtenues. Chacun de ces groupes est en effet composé d'un nombre suffisant de jeux d'essais pour pouvoir définir une tendance générale pour un algorithme. La moyenne globale est, quant à elle, réalisée sur plus de 140 jeux d'essais correspondant à des problèmes d'alignements différents. Obtenir de bons résultats sur la moyenne générale permet de mettre en évidence un algorithme d'alignement assez polyvalent.

5.8.2 Conditions expérimentales

Nous avons envisagé deux optiques différentes pour la réalisation des tests. La première consiste à tester l'algorithme avec différents paramètres, la seconde consiste à réaliser tous les alignements de nos jeux d'essai avec un seul paramétrage.

Tests avec variation des paramètres

Cette approche a pour objectif de déterminer pour chaque jeu d'essai le meilleur alignement que *Plasma II* est en mesure de réaliser. Cela revient donc à déterminer une borne supérieure pour notre algorithme sur les jeux d'essai de *Balibase*. L'idéal pour un algorithme en utilisant ce type de procédé est de pouvoir trouver pour chaque jeu d'essai au moins un paramètrage qui génère la solution de référence.

Les paramètres que nous avons faits varier sont les coûts d'ouverture et d'extension de brèches K et h, ainsi que la matrice de substitution $\mathcal{M}$. Pour éviter d'avoir à effectuer un nombre trop important de calculs sur chaque jeu d'essai, nous avons décidé de restreindre à quelques valeurs les variations possibles. Ceci n'est toutefois pas très gênant puisque l'on constate qu'à partir de certains seuils pour les valeurs de K et de h, il n'y a plus de modifications du résultat. Les variations pour ces paramètres correspondent à des intervalles de valeurs qui sont réalistes pour l'alignement de séquences. Ainsi les valeurs de K et h seront toujours définies négatives.

Même sur un intervalle donné, il n'est pas possible de tester toutes les valeurs. Nous avons donc subdivisé les valeurs de tests pour K et h. La valeur d'ouverture d'une brèche est généralement plus élevée que la valeur d'extension. Un paramètrage standard consiste à prendre les valeurs $K = -10$ et $h = -1$ pour un calcul d'alignement. Nous avons donc sélectionné nos intervalles autour de ces valeurs : $K \in [-8, -15]$ et $h \in [-0.5, -2]$.

Les premiers tests réalisés montraient de meilleurs résultats avec des matrices de substitution de type *Blosum* et *Gonnet*. Nous avons donc réalisé nos alignements en utilisant des matrices de ces deux familles. Là encore il ne nous a pas semblé judicieux de chercher à utiliser toutes les matrices de chacune de ces familles. Par exemple pour les matrices *Blosum*, nous avons sélectionné quelques matrices standard, allant de *Blosum* 40 à *Blosum*

350. En revanche, des matrices telles que *Blosum* 500 et *Blosum* 1000 nous ont semblé trop spécifiques pour être utilisées dans le cadre des alignements multiples sur les jeux d'essais de *Balibase*.

Tests avec paramètres fixés

Les tests avec un unique jeu de paramètres sont obligatoirement moins bons, mais ils offrent l'avantage de permettre de constater le comportement de *Plasma II* par rapport à la borne supérieure calculée avec paramètres variables. L'idéal pour l'algorithme est de pouvoir déterminer en fonction du jeu de séquences à aligner des paramètres qui soient adaptés.

Le jeu de paramètres que nous avons choisi est celui que l'on trouve le plus souvent dans les articles d'alignements multiples de séquences. Ils représentent en quelque sorte un jeu de paramètres standard, et nous l'avons utilisé ici comme le jeu par défaut pour *Plasma II*. Les valeurs de ces paramètres sont les suivantes :

- Matrice : *Gonnet 250*,
- $K = -10$,
- $h = -1$.

5.8.3 Résultats

Nous allons présenter ici les résultats obtenus pour les tests effectués dans les conditions données précédemment. Nous présentons tout d'abord les résultats avec variations des paramètres puis nous donnerons les résultats avec paramètres fixes.

Les évaluations qui sont données sont faites avec les fonctions de comparaison *SPS* et *CS* (section 3.8.1) de *Balibase* en prenant en compte pour chaque alignement le fichier d'annotation associé. À titre de comparaison, nous donnons également les valeurs obtenues par quelques uns des algorithmes les plus performants. Il s'agit de *Clustal W* et *T-Coffee* pour les plus anciens et de la dernière version de *MAFFT* [Katoh *et al.*, 2005], *Muscle* et *ProbCons* pour les plus récents. Ces trois derniers étant respectivement de 2005, 2004 et 2005. Pour ces cinq algorithmes nous avons également réalisé les tests avec les paramètres par défaut, et toutes les valeurs indiquées sont celles que nous obtenons.

Nous appelons *Plasma II std* (standard) les résultats obtenus avec le jeu de paramètres fixes et *Plasma II maxi* les meilleurs résultats obtenus avec les jeux de paramètres variables.

La moyenne est calculée en prenant en compte le nombre de jeux d'essais dans chaque référence. Les tests ont tous été réalisés sur le même ordinateur, et les temps indiqués correspondent à la durée pour réaliser l'ensemble des alignements de *Balibase*. Pour les résultats obtenus avec les jeux de paramètres variables, l'algorithme est utilisé plusieurs fois. Les temps de calculs pour *Plasma II maxi* ne sont donc pas indiqués.

Résultats avec le critère *SPS*

Pour le critère de comparaison *SPS* nous obtenons les résultats donnés dans le tableau 5.1. Les autres algorithmes donnés à titre de comparaison ont été utilisés avec leurs paramètres par défaut.

	Ref 1	Ref 2	Ref 3	Ref 4	Ref 5	Moyenne	Temps (sec)
Clustal W	0.809	0.932	0.723	0.834	0.858	0.828	120
T-Coffee	0.814	0.928	0.739	0.852	0.943	0.840	1653
MAFFT	0.829	0.931	0.812	**0.947**	**0.978**	0.867	98
Muscle	0.821	0.935	0.784	0.841	0.972	0.851	75
ProbCons	0.849	**0.943**	**0.817**	0.939	0.974	0.880	711
Plasma II std	0.832	0.916	0.729	0,823	0.918	0.844	343
Plasma II maxi	**0.898**	0.935	0.804	**0.947**	0.973	**0.906**	-

TAB. 5.1 – *Résultats avec le critère SPS.*

Résultats avec le critère *CS*

Pour le critère de comparaison *CS* nous obtenons les résultats donnés dans le tableau 5.2.

	Ref 1	Ref 2	Ref 3	Ref 4	Ref 5	Moyenne	Temps (sec)
Clustal W	0.707	0.592	0.481	0.623	0.634	0,656	120
T-Coffee	0.712	0.524	0.480	0.644	0.863	0,669	1653
MAFFT	0.736	0.525	0.595	0.822	**0.911**	0,712	98
Muscle	0.725	0.593	0.543	0.593	0.901	0,692	75
ProbCons	0.765	0.623	**0.631**	**0.828**	0.892	0,747	711
Plasma II std	0.734	0,555	0.350	0.784	0.768	0.679	343
Plasma II maxi	**0.850**	**0.679**	0.498	0.823	0.895	**0.794**	-

TAB. 5.2 – *Résultats avec le critère CS.*

5.8.4 Commentaires sur les résultats

Les résultats

Nous venons de présenter les résultats obtenus par *Plasma II* avec deux modes opératoires différents pour réaliser les tests. Nous allons maintenant comparer ces résultats avec ceux des autres algorithmes d'alignement multiple de séquences.

Avec le jeu de paramètres par défaut Les résultats obtenus par *Plasma II* avec le jeu de paramètres par défaut sont globalement bons, aussi bien pour le critère *SPS* que pour le critère *CS*. En effet, sur l'ensemble des jeux d'essais la moyenne des résultats de *Plasma II* est supérieure à la moyenne des résultats de *Clustal W*. Les résultats de *T-Coffee* sont très proches de ceux obtenus par *Plasma II*. En revanche les trois autres algorithmes donnent de meilleurs résultats, surtout *ProbCons* qui est très largement en tête.

De façon plus détaillée, nous constatons que la catégorie 1 est celle où *Plasma II* est le plus performant puisque les résultats sont même supérieurs à ceux de *Muscle*. L'algorithme semble donc plus performant pour l'alignement de séquences similaires. En revanche pour

la référence 3, les valeurs obtenues ne sont pas très bonnes, en particulier avec le critère *CS*.

Avec variation de paramètres Lorsque l'on fait varier les paramètres et que l'on ne considère que le meilleur alignement obtenu pour chaque jeu d'essais, les résultats sont bien meilleurs. En effet, la moyenne obtenue par *Plasma II* dans ces conditions est supérieure à celle de *ProbCons*. *ProbCons* donnant toutefois de meilleurs résultats sur la catégorie 3.

Ces résultats nous montrent que notre algorithme peut donner de meilleurs résultats en choisissant de bons paramètres d'alignement. La plupart des algorithmes déterminent par eux-même les paramètres qu'ils utilisent pour effectuer les calculs. Ce choix semble une option intéressante pour améliorer les performances de *Plasma II*.

Les temps d'exécution

Le temps d'exécution est bien sûr un aspect important, puisque pour être utilisable un algorithme doit être en mesure de fournir un résultat en un temps acceptable. Toutefois la rapidité n'est pas un facteur primordial pour le problème d'alignement multiple de séquences. Si l'algorithme est en mesure de fournir de bons résultats, le temps d'exécution devient secondaire dans la mesure où il reste raisonnable, c'est-à-dire quelques secondes. Pour comparer deux algorithmes, la qualité des résultats est beaucoup plus importante que les temps nécessaires pour les obtenir.

Les résultats obtenus expérimentalement nous permettent de déterminer des temps d'exécution de l'algorithme en fonction du nombre de séquences et de leurs longueurs. Nous constatons que pour tous les jeux d'essai de *Balibase*, les temps d'exécution sont rapides. Le jeu d'essai le plus important (1uky) est constitué de 24 séquences, et il faut environ une seconde pour obtenir le résultat.

Le temps total pour l'ensemble des jeux d'essais de *Balibase* permet de classer les différents algorithmes selon un critère de rapidité. *MAFFT* et *Muscle* sont les plus rapides, et *T-Coffee* est vingt fois plus lent. Nous pouvons constater que *Plasma II* est dans la moyenne, et que selon ce critère il ne peut rivaliser avec les plus rapides. Nous remarquons toutefois que *ProbCons*, qui donne les meilleurs résultats selon les critères *SPS* et *CS*, est deux fois plus lent que *Plasma II*.

Nous constatons que pour les temps d'exécution, le nombre de séquences joue un rôle plus important que leur longueur. Ce résultat peut être rapproché de la complexité que nous avions calculée pour l'algorithme. Pour n séquences à aligner, l'influence sur la complexité de l'algorithme est en effet en n^3.

5.9 Conclusion

Dans ce chapitre nous avons introduit les notions d'opérations d'édition étendues et d'alignement par blocs. Le bloc correspond à un alignement de séquences sur lequel sont définies des opérations spécifiques. Celle-ci permettent d'étendre de façon naturelle le concept d'opérations d'édition. Pour cela nous avons défini une opération de

match/mismatch, ainsi que des opérations d'insertion et de deletion pour les blocs. La restriction de ces opérations à des blocs composés d'une unique séquence, permet de retrouver les opérations d'édition pour deux séquences.

L'algorithme *Plasma II* que nous avons présenté est basé sur une extension du principe d'alignement par paire à l'aide des opérations d'édition étendues. Nous avons ainsi réalisé un algorithme d'alignement de deux blocs basé sur la programmation dynamique, et par extension un algorithme complet d'alignement multiple de séquences. L'algorithme peut être décomposé en plusieurs étapes. Tout d'abord une partie qui consiste à faire une pré-analyse des séquences en réalisant les alignements par paires afin de créer un *guide-tree*.

La partie principale, qui constitue le cœur de l'algorithme de *Plasma II*, sert à réaliser l'alignement de deux blocs. Cette partie est appelée itérativement sur l'ensemble des séquences initiales, suivant l'ordre établi par le *guide-tree*, jusqu'à obtenir un unique bloc contenant l'ensemble des séquences alignées.

Nous avons testé notre algorithme avec les jeux d'essais proposés dans *Balibase*, pour cela nous avons défini deux modes opératoires différents : l'un avec un jeu de paramètres par défaut et l'autre en choisissant le meilleur résultat obtenu avec une variation des paramètres. Dans les deux cas, les résultats sont de bonne qualité :

- *Plasma II std* obtient en moyenne pour les deux critères de meilleurs résultats que *Clustal W* et *T-Coffee*, mais est moins performant que *MAFFT*, *Muscle* et *ProbCons*.
- *Plasma II maxi* obtient en revanche de meilleurs résultats que les autres algorithmes, avec une différence assez marquée pour le critère *CS*.

Nous constatons que dans sa version standard, l'algorithme de *Plasma II* possède une marge de progression assez importante : $7,3\%$ pour le critère *SPS* et $16,9\%$ pour le critère *CS*. Une étude du choix des paramètres utilisés pour réaliser les alignements semble très intéressante à mener.

Dans le chapitre suivant nous présentons quelques modifications apportées à notre algorithme avec pour objectif d'améliorer ses résultats.

Chapitre 6

Variantes basées sur l'algorithme de *Plasma II*

6.1 Introduction

L'objectif que nous poursuivons consiste à faire de *Plasma II* une boîte à outils pour l'alignement multiple de séquences. Cela sous-entend de proposer un maximum de variations possibles autour de l'algorithme d'alignement par blocs, laissant ainsi à l'utilisateur le choix des fonctionnalités qu'il souhaite avoir pour le calcul du résultat qu'il cherche. En effet, selon la catégorie à laquelle appartiennent les séquences il est souvent préférable de pouvoir adapter les paramètres utilisés pour réaliser l'alignement. L'utilisateur peut également se rendre compte après un premier alignement que les paramètres employés n'étaient certainement pas les meilleurs, il faut donc lui laisser la possibilité de les modifier pour refaire l'alignement.

Une autre optique, plus avancée celle-là, consiste à déterminer un jeu de paramètres pour chaque type d'alignement. Une analyse des séquences de chacun des deux blocs à aligner peut permettre de déterminer les paramètres qui semblent les plus adaptés. Ainsi, le choix des paramètres d'alignement utilisés avec la fonction de somme des paires pondérée peut être adapté à chaque paire de séquences en fonction de leur similarité. La possibilité d'adapter les paramètres d'alignement peut également être mise en place en utilisant des fonctions d'évaluation telles que *T-Coffee* ou la fonction de *Coffee*. Ces deux fonctions prennent en effet les séquences initiales, afin de créer une évaluation plus adaptée. Elles sont obtenues à partir des cas simples combinant les différents alignements de deux ou trois séquences.

Au chapitre précédent, nous avons cherché à tester notre implémentation de *Plasma II* sur *Balibase*, car avec la plupart des jeux d'essais proposés est fourni un alignement de référence. En comparant attentivement nos résultats avec cette référence, nous avons décelé certaines erreurs d'alignements fréquentes. Nous avons donc cherché à les répertorier et à les corriger. La méthode utilisée pour calculer la valeur de l'alignement multiple ne peut pas être modifiée, les solutions proposées consituent en quelque sorte une "réparation" de l'alignement. Pour cela, il est nécessaire de réaliser une opération que l'on peut qualifier

de post-alignement. Celle-ci a pour objectif de répertorier les positions où l'alignement ne semble pas satisfaisant. Chaque zone ainsi répertoriée doit pouvoir être prise séparément, et corrigée au mieux.

Bon nombre de choix faits pour l'algorithme présenté au chapitre précédent peuvent être adaptés. Nous avons par exemple choisi d'office la fonction de somme des paires pondérées pour évaluer la qualité de nos alignements par blocs. Cette fonction est souvent utilisée car en patique elle est assez simple à mettre en œuvre, et elle offre des résultats satisfaisants dans bien des cas. Nous avons ajouté à notre version de *Plasma II* la possibilité de pénaliser différemment les brèches de début et de fin d'alignement afin de déterminer l'impact que ces paramètres peuvent avoir sur la qualité de l'alignement. D'autres fonctions ont également été définies afin d'évaluer la qualité des alignements. Nous proposons ici une version de l'algorithme de *Plasma II* avec la fonction d'évaluation de *Coffee*. Plusieurs algorithmes existants proposent de réaliser des alignements successifs. L'objectif de cette méthode est d'affiner le *guide-tree* en utilisant chaque alignement multiple intermédiaire pour créer un nouvel arbre. Enfin, nous proposons une méthode de correction d'alignement. Il ne s'agit pas ici d'intervenir sur les calculs lors de la construction du résultat, mais plutôt de déterminer s'il est possible d'améliorer ce résultat.

6.2 Utilisation de coûts différents pour les brèches de début et fin d'alignement

6.2.1 Introduction

Dans la version de *Plasma II* que nous avons présentée au chapitre précédent, tous les coûts de brèches étaient calculés de la même façon, quel que soit leur emplacement dans l'alignement résultant. Toutefois, il convient de prendre en compte un facteur particulier pour cette évaluation. En effet, les séquences qui doivent être alignées sont des fragments qui ont été artificiellement coupés. Chercher à évaluer les brèches de début et fin d'alignement revient à dire qu'elles ont un sens du point de vue de l'évolution, au même titre que les autres brèches insérées dans l'alignement.

Nous allons donc présenter ici une modification de l'algorithme de *Plasma II* permettant de considérer un coût différent pour les brèches de début et de fin d'alignement. Nous commençons pour cela par présenter un cas particulier qui consiste à attribuer un coût nul à ces brèches. Car de par sa nature, ce cas a l'avantage de pouvoir être traité au moyen d'une implémentation simplifiée. Dans un deuxième temps nous proposerons une généralisation avec un modèle de brèches à coût affine.

6.2.2 Présentation des deux cas

Brèches à coût nul

Associer un coût nul à une brèche revient à l'ignorer totalement pour le calcul de la somme des paires pondérées. Soient S_i et S_j deux des séquences d'un alignement multiple de longueur l. Soient d_i et d_j les longueurs des brèches de débuts des séquences S_i et S_j. Et

soient f_i et f_j les longueurs des brèches de fin de S_i et S_j. Nous appelons $d = \max(d_i, d_j)$ et $f = \max(f_i, f_j)$. L'évaluation associée à S_i et S_j est à réaliser entre les positions d et $l - f$.

Brèches à coût non nul

Associer un coût nul permet de ne pas prendre de décision quant aux parties coupées de la séquence. Toutefois cela offre une certaine liberté pour la position de début et de fin des séquences. Dans le cas de séquences similaires, l'impact que cela peut avoir est assez limité. En revanche, pour des séquences assez différentes, les brèches internes de l'alignement risquent d'être repoussées en début et fin de séquences.

Nous avons donc décidé d'implémenter également dans *Plasma II* une évaluation non nulle des brèches de début et de fin de séquences. Pour cela nous ajoutons deux paramètres supplémentaires à notre algorithme, à savoir un coût K' de création et un coût h' d'extension des brèches. Le coût associé à une brèche de longueur l positionnée en début ou fin de séquence est donc de la forme : $K' + l.h'$. En prenant des valeurs identiques pour K et K' ainsi que pour h et h', nous retrouvons l'évaluation vue au chapitre V.

6.2.3 Implémentation dans *Plasma II*

Brèches à coût nul

Pour une évaluation nulle des brèches de début et de fin, l'implémentation à réaliser est assez simple. Nous devons pour cela différencier les deux types de brèches.

Brèches de début de séquence : Pour ce cas, l'implémentation est semblable au cas standard vu au chapitre précédent. En effet, obtenir un coût nul pour les brèches de début de séquences peut être fait aisément lors de l'initialisation des valeurs dans la première ligne et la première colonne des matrices. En effet pour les matrices où ces valeurs ont un sens, il suffit de prendre l'initialisation des brèches avec une valeur nulle, car les valeurs de la première colonne et de la première ligne représentent le coût engendré par le décalage d'un bloc par rapport à l'autre.

Brèches de fin de séquence : Pour les brèches de fin d'alignement, il n'est pas possible de procéder de la même façon. Nous utilisons la méthode généralement employée, qui consiste à rechercher le maximum sur la dernière ligne et la dernière colonne. Une fois cette valeur $v_{max} = M(i_{max}, j_{max})$ déterminée, elle est prise comme valeur pour l'évaluation de l'alignement des deux blocs.

- Si v_{max} est obtenue sur la dernière ligne alors nous avons $i_{max} = m$. Toutes les valeurs de M de cette ligne placées entre $M(m, j_{max})$ et $M(m, n)$ prennent pour valeur v_{max}.
- Si v_{max} est obtenue sur la dernière colonne alors nous avons $j_{max} = n$. Toutes les valeurs de M de cette ligne placées entre $M(i_{max}, n)$ et $M(m, n)$ prennent pour valeur v_{max}.

De cette façon, on associe le coût au maximum de la dernière ligne et de la dernière colonne, puis la brèche qui le suit se voit attribuer une valeur nulle. En effet, la construction progresse dans la matrice M à partir de $M(m, n)$ jusqu'à $M(i_{max}, j_{max})$, créant ainsi la brèche.

Brèches à coût quelconque

La méthode utilisée pour les brèches à coût nul n'est plus utilisable dans ce cas. Il faut utiliser la même méthode que celle utilisée par le reste de l'algorithme, mais ici avec des coûts de création et d'extension de brèches différents.

Brèches de début de séquence : La prise en compte des brèches de début de séquences se fait lors de l'initialisation des différentes matrices, pour lesquelles cela a un sens. Les calculs sont les mêmes que ceux de l'algorithme de base, mais avec les valeurs K' et h'. Cela permet donc de remplir les premières lignes et colonne de ces matrices.

Brèches de fin de séquence : La prise en compte des brèches de fin avec un coût non nul oblige à chaque calcul de l'algorithme de vérifier si l'on est positionné sur la dernière ligne ou sur la dernière colonne.

En pratique pour l'algorithme, il suffit d'utiliser la méthode classique sur les $m - 1$ premières lignes et $n - 1$ premières colonnes. La dernière ligne et la dernière colonne sont calculées toutes les deux à la fin en modifiant l'algorithme. Le coût des brèches est différent pour les insertions de la dernière ligne ainsi que pour les deletions de la dernière colonne.

6.2.4 Résultats

Nous avons testé cette nouvelle implémentation de *Plasma II* sur les jeux d'essais de *Balibase*. L'objectif est ici de déterminer l'impact que peut avoir un coût différent pour les brèches de début et fin d'alignement. Pour cela, nous avons évalué notre algorithme dans les deux cas cités précédemment :

- avec des brèches de début et fin à coût réduit. Pour cela nous avons utilisé les paramètres $K' = K/2$ et $h' = h/2$, ces valeurs représentant un cas intermédiaire pour K' et h'. Les résultats obtenus sont donnés dans le tableau 6.1,
- avec des brèches de début et fin à coût nul. Les résultats obtenus sont présentés dans le tableau 6.2

Les calculs ont été effectués dans les mêmes conditions qu'au chapitre précédent, c'est-à-dire $K = -10$, $h = -1$ et la matrice *Gonnet 250*. Les résultats obtenus sont donnés avec les deux critères SPS et CS. Nous redonnons également les valeurs obtenues au chapitre précédent (*Plasma II std*) afin de pouvoir faire une comparaison avec les nouveaux résultats (*Plasma II gap*). Nous avons décomposé les jeux de la Référence 1 en fonction du pourcentage de similarité des séquences composant les jeux d'essais afin de mieux comprendre l'influence du nouvel algorithme en fonction des similarités. Nous donnons donc les résultats pour les trois sous-catégories suivantes :

- moins de 20% de similarité,

– entre 20% et 40% de similarité,
– plus de 40%.

	Critère SPS		Critère CS	
	Plasma II std	*Plasma II gap*	*Plasma II std*	*Plasma II gap*
Référence 1				
$< 20\%$	0.597	0.597	0.395	0.395
$< 40\%$	0.905	0.905	0.842	0.842
$> 40\%$	0.945	0.958	0.893	0.921
Moyenne	0.832	0.835	0.734	0.741
Référence 2	0.916	0.911	0.555	0.529
Référence 3	0.729	0.712	0.350	0.341
Référence 4	0.823	0.686	0.784	0.310
Référence 5	0.918	0.785	0.768	0.519
Moyenne	0.844	0.821	0.679	0.619

TAB. 6.1 – *Résultats pour* $K' = K/2$ *et* $h' = h/2$.

	Critère SPS		Critère CS	
	Plasma II std	*Plasma II gap*	*Plasma II std*	*Plasma II gap*
Référence 1				
$< 20\%$	0.597	0.635	0.395	0.460
$< 40\%$	0.905	0.905	0.842	0.846
$> 40\%$	0.945	0.961	0.893	0.927
Moyenne	0.832	0.849	0.734	0.766
Référence 2	0.916	0.911	0.555	0.522
Référence 3	0.729	0.668	0.350	0.325
Référence 4	0.823	0.584	0.784	0.214
Référence 5	0.918	0.807	0.768	0.604
Moyenne	0.844	0.819	0.679	0.631

TAB. 6.2 – *Résultats pour* $K' = 0$ *et* $h' = 0$.

Pour $K' = K/2$ et $h' = h/2$ nous obtenons les mêmes tendances aussi bien avec le critère SPS que le critère CS. Nous constatons une amélioration uniquement pour les jeux de séquences les plus similaires de la référence 1. Pour les autres références, il y a une détérioration des résultats, en particulier pour les références 4 et 5. La moyenne globale est au final moins bonne qu'avec l'algorithme classique du chapitre précédent.

Pour $K' = 0$ et $h' = 0$ la tendance du cas précédent s'accentue. Pour la Référence 1 les résultats sont encore meilleurs. En effet, la moyenne obtenue pour le critère SPS est égale à la moyenne *ProbCons*. Pour le critère CS la valeur moyenne obtenue par *Plasma II* est très légèrement supérieure. Pour les autres références, les résultats sont en revanches moins bons qu'avec l'algorithme standard.

Même si en moyenne les résultats sont moins bons, cette méthode peut tout de même être intéressante. En effet il est assez simple de déterminer si un jeu de séquences appartient à la référence 1. Ainsi, chercher à savoir si les séquences à aligner sont suffisamment similaires avant de réaliser l'alignement multiple semble donc une modification intéressante pour l'algorithme de *Plasma II*. Si tel est le cas il est préférable de ne pas pénaliser les brèches de début et de fin d'alignement. En prenant ces résultats pour la référence 1 et ceux obtenus par l'algorithme *Plasma II* du chapitre précédent pour les autres références nous obtenons les moyennes suivantes :

- 0,853 pour le critère SPS,
- 0,697 pour le critère CS.

Nous rappelons que les valeurs obtenues par Muscle sont respectivement 0,851 et 0,692. Utiliser un coût nul pour les brèches de début et de fin d'alignement avec les jeux d'essais de la référence 1 permet donc d'obtenir des résultats de meilleure qualité que ceux de Muscle.

6.2.5 Conclusion

Nous avons présenté dans cette section une méthode permettant de moins pénaliser les brèches de début et de fin d'alignement. Pour cela il suffit de modifier l'initialisation des matrices ainsi que la méthode de calcul de la dernière ligne et de la dernière colonne.

Pour déterminer l'influence que cela peut avoir sur la qualité des alignements nous avons de nouveau utilisé les jeux d'essais de *Balibase*. Nous constatons au vu des résultats expérimentaux que cette méthode améliore la qualité des alignements sur les jeux d'essais les plus similaires de la référence 1. Pour des brèches de début et de fin à coût nul, notre algorithme parvient même à égaler *ProbCons*. En revanche pour les autres références, les résultats obtenus sont de moins bonne qualité que ceux présentés au chapitre 5. En particulier pour la référence 4 lorsque $K = h = 0$ où les valeurs sont très mauvaises.

Utiliser cette méthode uniquement sur les jeux d'essais de la référence 1 permet d'augmenter de façon significative la moyenne des résultats. L'algorithme devient ainsi meilleur que Muscle, et se classe derrière MAFFT et ProbCons.

6.3 Implémentation de la fonction d'évaluation de *Coffee* dans *Plasma II*

6.3.1 Rappels sur la fonction d'évaluation de *Coffee*

Principe

Le principe de la fonction d'évaluation utilisée dans *Coffee* [Notredame *et al.*, 1998] est simple. Soit n séquences à aligner, dans un premier temps les $n.(n-1)$ alignements par paires de toutes les séquences sont réalisés. Tous les alignements sont parcourus, et chacune des paires de résidus ainsi obtenues sont conservées. Ces paires vont être ensuite fréquemment utilisées pour évaluer la qualité d'un alignement multiple, il est donc important de les conserver dans une structure rendant facile leur accès.

Pour évaluer un alignement multiple, il est nécessaire de parcourir deux par deux toutes les séquences qui le constitue. Il reste à comparer ces paires de séquences avec les alignements par paires initiaux. L'évaluation se fait en dénombrant les paires de résidus communes aux couples de séquences de l'alignement multiple et aux alignements par paires.

Ce principe se base sur l'hypothèse que le calcul des alignements par paire donne de bons résultats car il est basé sur un algorithme exact. La fonction prend donc les alignements par paire comme référence et les compare à l'alignement multiple. Un alignement multiple est d'autant meilleur que chacune des paires de séquences qui le constitue sont similaires aux alignements par paires de référence.

$$\text{Coffee} = \frac{\sum \textit{ paires de residus communes}}{\sum \textit{ paires de residus}}$$

La valeur obtenue indique le pourcentage de similarité entre l'alignement multiple et les alignements par paire de référence.

Complexité

Nous allons décomposer le calcul en trois parties distinctes pour déterminer la complexité générale de la fonction :

- Les alignements par paires réalisés dans la première phase de l'évaluation sont les mêmes que ceux permettant d'obtenir la matrice des distances avec l'algorithme classique utilisé pour la fonction de somme des paires. On obtient donc pour n séquences de longueur au plus l une complexité en $O(n^2.l^2)$.
- Pour conserver chaque paire de résidus, il suffit de parcourir une fois tous les alignements par paire, on obtient une complexité en $O(n.(n-1).l/2)$, c'est-à-dire $O(n^2.l)$.
- Si les paires de résidus sont conservées dans des structures appropriées, il est possible d'y faire référence en temps constant au moyen d'un accès direct. L'évaluation de l'alignement multiple consiste à déterminer pour chaque paire de résidus si elle était présente dans l'alignement par paire correspondant. Le parcours d'une paire de séquences a une complexité de $O(l)$, donc pour l'alignement multiple on obtient de nouveau une complexité en $O(n^2.l)$.

La complexité de la fonction d'évaluation de *Coffee* pour un alignement multiple de n séquence de longueur l est donnée par :

$$O(n^2.l^2 + n^2.l + n^2.l)$$

C'est-à-dire, en supprimant les quantités négligeables :

$$O(n^2.l^2)$$

A titre de comparaison, il est intéressant de remarquer que la partie proprement dite d'évaluation de *Coffee* a la même complexité que le calcul de la somme des paires. La somme des paires pondérée nécessite quant à elle le calcul du *guide-tree* et donc de la matrice des distances. La somme des paires pondérée et la fonction de *Coffee* ont donc la même complexité.

6.3.2 Implémentation dans *Plasma II*

Différences avec l'algorithme précédent

L'utilisation de la somme des paires pondérée dans *Plasma II* est intégrée directement au cœur de l'algorithme réalisant le calcul pour l'alignement de deux blocs. Notre implémentation propose ici une méthode qui remplace cette fonction par celle de *Coffee*. Plusieurs choses sont à modifier dans l'algorithme. Certaines demandent de nouvelles opérations ou de nouvelles structures de stockage :

- Déterminer toutes les paires de résidus initiales, et les conserver de façon à pouvoir faire en temps constant l'évaluation de chaque paire de résidus lors de l'alignement de deux blocs.
- Permettre un calcul simple de la fonction lors des calculs dans les matrices. Le problème qui se pose par rapport à la somme des paires provient de la nature de la fonction. Il s'agit d'un quotient, et l'additivité n'est donc pas possible à partir de la valeur d'un sous-problème.

En revanche, il y a également de nombreuses simplifications par rapport à l'algorithme présenté au chapitre précédent. Dans *Coffee*, les brèches sont comptabilisées dans la partie des alignements par paires. En revanche pour l'alignement multiple elles ne sont plus prises en compte. Comme la fonction cherche à replacer les mêmes paires de résidus que pour les alignements par paires, les brèches se retrouvent de façon naturelle également aux mêmes places. Il n'est donc plus nécessaire de définir toutes les matrices que nous avons utilisées précédemment, car il n'y a plus de différences à faire entre les ouvertures et les extensions de brèche. L'implémentation de la fonction de *Coffee* peut être faite simplement en utilisant le même principe que pour l'alignement à coût de brèches constant.

Il convient de remarquer que même si le coût des brèches et la matrice de substitution ne font plus partie des paramètres pour l'alignement de deux blocs, ils sont toujours utilisés dans la partie initiale pour générer les alignements par paires.

Implémentation

L'implémentation de la fonction d'évaluation de *Coffee* est plus simple que celle de la somme des paires pondérées avec coût de brèches affine. Nous allons tout d'abord voir comment il convient de la mettre en place pour conserver les paires de résidus. Nous expliciterons ensuite comment réaliser l'alignement par blocs avec cette fonction, et plus particulièrement la différence de calcul en raison de la recherche du maximum d'un quotient.

Recherche des paires de résidus : Avec la fonction de somme des paires, les alignements par paire sont utilisés uniquement pour déterminer le *guide-tree*. Il n'est donc pas forcément nécessaire de chercher à obtenir des alignements par paires de très bonne qualité. En revanche, l'évaluation d'un alignement multiple avec la fonction de *Coffee* dépend entièrement de ce prétraitement, il est donc important d'obtenir des alignements par paires de bonne qualité. Pour cela nous avons choisi de réaliser des alignements par paires avec coût de brèches affine.

Comme une faible variation des valeurs de paramètres n'entraîne que rarement une variation de l'alignement par paire, des paramètres standard peuvent être choisis par défaut. Il est également possible de déterminer des paramètres "optimaux" pour chaque couple de séquences en fonction de leur similarié.

Dans notre implémentation, nous cherchons à optimiser les accès aux paires de résidus. Le but est de déterminer en temps constant si une paire de résidus de l'alignement multiple est présente ou non dans l'alignement par paire correspondant. Nous utilisons pour cela deux tableaux, chacun représentant une des deux séquences de l'alignement par paire. Par exemple pour deux séquences S_1 de longueur n_1 et S_2 de longueur n_2, soit le tableau T_1 représentant la séquence S_1, et soit le tableau T_2 représentant la séquence S_2. La longueur de chacun des deux tableaux est égale à la longueur de la séquence qu'il représente. Dans chacune des cases, on indique la paire de résidus déterminée lors de l'alignement par paire de S_1 avec S_2. Deux cas sont alors possibles :

- Soit α le $i^{\text{ème}}$ caractère de S_1. Si α est aligné avec une brèche dans S_2, alors on a $T_1[i] = -1$,
- En revanche, si α est aligné avec β, $j^{\text{ème}}$ caractère de la séquence S_2, alors on obtient $T_1[i] = j$.

Même si de cette façon les informations sont conservées en double pour chaque paire de résidus, pour deux séquences données, il devient immédiat de savoir si deux caractères de ces séquences forment une paire de résidus dans l'alignement initial.

Pour évaluer un alignement multiple de $n > 2$ séquences, il faut donc conserver les positions des paires de résidus en double pour chacun des $n.(n-1)/2$ alignements par paires. C'est-à-dire $n.(n-1)$ tableaux. La complexité de cette opération est négligeable devant le calcul d'alignement par bloc. La mémoire nécessaire pour conserver ces informations n'est pas non plus importante pour un nombre de séquences raisonnable. Par exemple, pour des séquences de longueur moyenne 300, la conservation des paires de résidus nécessite la mémoire suivante :

- 50 Ko pour 5 séquences,
- 1 Mo pour 20 séquences,
- 25 Mo pour 100 séquences.

Ces valeurs permettent de justifier le choix que nous avons fait puisque la mémoire nécessaire est faible par rapport à ce qui est maintenant disponible sur les ordinateurs.

Alignement avec la fonction de *Coffee* : Comme indiqué précédemment, l'utilisation de cette fonction nécessite quelques modifications dans l'algorithme de *Plasma II* tel qu'il a déjà été présenté. En effet l'évaluation d'un alignement avec cette fonction nécessite de calculer le quotient de deux nombres. Il n'est donc plus possible d'utiliser directement l'additivité pour calculer une valeur à partir de ses sous-problèmes.

6.3.3 Résultats

Là encore, nous avons testé cette version modifiée de notre algorithme avec les jeux d'essais de *Balibase*. Pour pouvoir comparer les résultats de cette version de l'algorithme nous avons de nouveau utilisé les paramètres par défaut du chapitre 5.

Afin de déterminer en détail le comportement de l'algorithme, nous donnons de nouveau les résultats pour les trois sous-catégories de la référence 1.

Nous présentons dans le tableau 6.3 les résultats obtenus avec la fonction de *Coffee*, et à titre de comparaison nous redonnons les résultats obtenus par la méthode de somme de paires pondérée.

	Critère SPS		Critère CS	
	Plasma II	*Plasma II*_C	*Plasma II*	*Plasma II*_C
Référence 1				
$< 20\%$	0.597	0.476	0.395	0.287
$< 40\%$	0.905	0.929	0.842	0.875
$> 40\%$	0.945	0.975	0.893	0.949
Moyenne	0.832	0.818	0.734	0.735
Référence 2	0.916	0.911	0.555	0.463
Référence 3	0.729	0.700	0.350	0.346
Référence 4	0.823	0.847	0.784	0.688
Référence 5	0.918	0.888	0.768	0.705
Moyenne	0.844	0.831	0.679	0.651

TAB. 6.3 – *Résultats avec les critères SPS et CS.*

Les résultats obtenus en moyenne sont moins bons pour les deux critères SPS et CS. Toutefois en regardant les valeurs plus en détails nous constatons qu'il ne s'agit pas là d'une tendance générale sur l'ensemble des jeux d'essais.

Pour la référence 1, les résultats obtenus étaient prévisibles. En effet l'alignement multiple de plusieurs séquences très similaires permet d'obtenir des projections très proches des différents alignements de deux séquences. Or la fonction d'évaluation que nous utilisons est basée sur les alignements de tous les couples de séquences. Ainsi nous constatons que pour les jeux de séquences les plus similaires les résultats sont améliorés. En revanche pour les similarités les plus faibles les résultats sont beaucoup moins bons.

Pour les autres résultats les variations ne sont pas très importantes, mais seule la référence 4 obtient une meilleure moyenne.

6.3.4 Conclusion

Dans cette partie nous avons présenté l'implémentation de la fonction de *Coffee* dans notre algorithme, en donnant en particulier les détails permettant d'optimiser les calculs. Les résultats obtenus en utilisant cette modification dans *Plasma II* sont moins bons que ceux que nous obtenons en utilisant la fonction de somme des paires pondérée. Seuls les jeux d'essais composés de séquences similaires et la référence 4 obtiennent de meilleurs résultats.

Le défaut de la fonction de *Coffee* est que, pour un résidu donné dans une séquence, elle n'associe qu'un seul résidu dans une autre séquence. Or associer une valeur nulle dans tous les autres cas est assez pénalisant puisque l'on ne prend pas en considération la probabilité de trouver ces deux résidus alignés.

Utiliser *T-Coffee* comme fonction d'évaluation nous semblait une alternative intéressante. En effet cette fonction ne se focalise pas uniquement sur les alignements globaux de tous les couples de séquences. Ainsi, pour un résidu donné dans une séquence peuvent être associés plusieurs résidus dans une autre séquence. Toutefois la description qui est donnée dans l'article de *T-Coffee* ne permet pas de comprendre exactement son implémentation. Ainsi rien n'est dit sur l'utilisation de la bibliothèque d'alignements pour l'évaluation de l'alignement multiple.

6.4 Alignement après construction d'un nouveau *guide-tree*

6.4.1 Principe des alignements successifs

Comme nous l'avons vu dans la première partie de ce manuscrit, le *guide-tree* obtenu à partir de tous les alignements par paires est de bonne qualité. Toutefois l'alignement multiple obtenu à partir de ce *guide-tree* peut être utilisé pour créer un nouveau *guide-tree*. Cette méthode a été présentée la première fois dans l'algorithme *Multalin* [Corpet, 1988].

On peut définir facilement une opération de projection d'un alignement A de n séquences vers un alignement A' de $m < n$ séquences. A' correspond à m séquences de A auxquelles ont été supprimées toutes les colonnes constituées uniquement de brèches.

La projection d'un alignement multiple A suivant tous les couples de séquences (i, j) permet d'obtenir tous les alignements de deux séquences A_{ij}. Ces alignements sont plus pertinents que les alignements de deux séquences standard car ayant été obtenus par calculs avec toutes les séquences. La matrice des distances qu'ils permettent de calculer, ainsi que le *guide-tree* qui en découle sont également de meilleure qualité.

Ce nouvel arbre s'il est différent du premier permet donc de calculer un nouvel alignement multiple, théoriquement meilleur que le premier. Ce processus peut être réitéré un nombre fixé de fois ou jusqu'à stabilisation du résultat.

6.4.2 Implémentation dans *Plasma II*

Nous avons implémenté dans *Plasma II* le principe des alignements successifs. Une des principales difficultés est de déterminer quand l'algorithme doit s'arrêter.

La cas simple d'arrêt correspond au cas où deux arbres successifs sont identiques. Il suffit donc d'implémenter une méthode de comparaison de deux arbres. Toutefois, même si le nombre d'arbres possibles est fini, rien ne nous permet de dire a priori pour un problème donné qu'il n'y aura pas de cycle au niveau de l'apparition des arbres. Or l'objectif dans cette section est de déterminer le comportement de notre algorithme lorsque l'on réitère le processus d'alignement d'un nouveau *guide-tree*. C'est pourquoi nous avons implémenté cette méthode uniquement sur un nombre fini d'itérations. Nous ajoutons donc à notre algorithme un paramètre correspondant au nombre de *guide-tree* devant être calculés. Une valeur de 1 pour ce paramètre permet donc de revenir au cas classique du chapitre précédent.

6.4.3 Tests sur les jeux de séquences de *Balibase*

Conditions expérimentales

Les jeux d'essais sont toujours ceux de *Balibase*. Les résultats ont été obtenus avec les mêmes paramètres qu'au chapitre 5 : $K = -10$, $h = -1$ et la matrice *Gonnet 250*. Nous avons choisi de faire tous les tests sur les jeux d'essais de *Balibase* en calculant à chaque fois un nombre fixé de *guide-tree*.

Résultats

Première itération : Avec le calcul d'un nouveau *guide-tree*, nous constatons que les résultats sont exactement les mêmes pour tous les jeux d'essai. Le second *guide-tree* est toutefois différent du premier pour tous les exemples, mais principalement au niveau des longueurs de branches. Cependant ces nouvelles valeurs ne modifient nullement les résultats.

Itérations suivantes : L'alignement multiple obtenu avec le second *guide-tree* étant le même qu'avec le premier, les itérations suivantes donnent bien entendu toujours les mêmes résultats.

6.4.4 Conclusion

Nous avons cherché à implémenter la stratégie de réalignement par calculs successifs des *guide-trees*. Notre programme pouvant ainsi calculer un nombre de *guide-tree* donné en paramètre. Afin de déterminer l'influence de cette méthode nous avons utilisé les jeux d'essais de *Balibase*.

Les résultats obtenus sont surprenant puisque cette méthode ne nous permet pas d'obtenir d'autres alignements multiples. Même si pour chaque jeu le second *guide-tree* est différent, l'alignement multiple résultant n'est pas modifié. Ce constat nous permet de dire que le premier *guide-tree* obtenu à partir des distances entre toutes les paires de séquences est déjà de bonne qualité.

6.5 Optimisation du résultat par correction des erreurs d'alignements

6.5.1 Introduction

Les méthodes présentées jusqu'ici dans ce chapitre avaient pour objectif d'améliorer la qualité de l'alignement multiple pendant la phase de calculs. Nous avons également cherché à implémenter une méthode pour améliorer un alignement. Ainsi ce nouvel algorithme sert-il de post-optimisation pour le résultat obtenu par *Plasma II* ou par tout autre algorithme d'alignement multiple.

L'objectif est ici de chercher à faire varier la position des brèches afin d'essayer de les replacer au mieux. Une première idée pourrait être de faire varier les brèches sur toute la

longueur de l'alignement, et être ainsi certain de les repositionner au mieux. Toutefois une telle stratégie est d'autant plus coûteuse en temps que les séquences à aligner sont longues et nombreuses. De plus cela revient à dire que l'alignement préalablement effectué n'a pas été utile, et n'a pas permis de mettre en évidence des zones que l'on peut considérer comme 'bien alignées'. L'insertion d'une ou plusieurs brèches dans une telle partie de l'alignement ayant pour effet de détériorer grandement la qualité de l'alignement. Le problème ici va être de déterminer un moyen de différencier au mieux les parties que l'on peut considérer comme étant 'bien alignées' des parties 'mal alignées'. Nous appellerons zone un sous-alignement du bloc que l'on cherche à améliorer, formé de toutes les sous-séquences de ce bloc entre deux positions données. Une zone mal alignée sera une zone contenant au moins une brèche.

Plusieurs méthodes s'offrent à nous pour améliorer la qualité d'une zone mal alignée. La plus simple consiste à faire varier la position de chaque brèche sur toute la longueur de cette zone. Avec un simple algorithme de descente on peut ainsi de proche en proche repositionner au mieux chaque brèche. Pour ne pas dégrader la solution nous proposons également de repositionner simultanément les brèches contenues dans plusieurs séquences ayant des positions initiales et finales identiques.

La difficulté principale consiste à trouver une fonction d'évaluation qui soit en mesure de dire que la partie mal alignée obtenue par *Plasma II* est de moins bonne qualité que celle de *Balibase*.

6.5.2 Présentation des erreurs à corriger

En comparant les alignements multiples obtenus par *Plasma II* avec les alignements optimaux de *Balibase*, nous avons constaté que plusieurs problèmes étaient récurrents. Nous exposons ici la liste des principales erreurs que nous avons détectées et auxquelles notre algorithme doit pouvoir apporter une solution :

Brèches morcelées

L'algorithme crée une alternance de brèches et de lettres dans les séquences. Ce morcellement des séquences provoque un allongement de l'alignement final (environ 5 à 10%), ainsi qu'une baisse de qualité. Chaque partie de l'alignement où un morcellement se produit ressemble à l'exemple de la figure 6.1. Les brèches sont insérées dans toutes les séquences, sauf une. La présence de trop nombreuses brèches rend difficile un réarrangement. En effet, comme le montre la longueur excessive de l'alignement, il ne s'agit pas ici uniquement d'une redistribution des brèches, il faut également être en mesure de pouvoir en supprimer une partie.

Brèches isolées.

Le problème consiste ici à déterminer la dimension de la zone sur laquelle il faut essayer de replacer au mieux une brèche isolée, dans un alignement. Quelle marge de déplacement faut-il accorder à la brèche, et donc quelle dimension doit-on donner à la zone mal alignée ?

```
RDKAD-F-G-A-I-N-KLE
RAKED-H-S-A-I-L-LNE
HGVEVEVLLRHLAALPQDI
EGKSL-E-E-I-I-R-SSE
```

FIG. 6.1 – Exemple de brèches morcelées

```
RDKADFGAIN-------KLE          R-------DKADFGAINKLE
RAKEDHSAIL-------LNE          R-------AKEDHSAILLNE
HGVEVEVLLRHLAALPQDIQ          HGVEVEVLLRHLAALPQDIQ
EGKSLEEIIR-------SSE          E-------GKSLEEIIRSSE
 Résultat Plasma II                    Référence
```

FIG. 6.2 – Exemple de brèches isolées à déplacer.

Le même problème s'applique à un ensemble de brèches superposées, comme dans l'exemple de la figure 6.2.

Brèches voisines.

Le problème ici consiste à déterminer une distance à partir de laquelle il faut considérer que deux brèches isolées doivent appartenir à une même zone mal alignée. Il faut pouvoir définir une distance entre brèches en dessous de laquelle il semble préférable de regrouper ces brèches. Ainsi dans l'exemple présenté à la figure 6.3, les deux brèches doivent être regroupées en une seule.

Fractionnement des brèches.

Une ou plusieurs brèches d'une zone mal alignée doivent pouvoir être fractionnées. L'algorithme doit ici être en mesure de résoudre le problème inverse de celui des brèches voisines. Ce problème est le plus difficile de ceux que nous venons d'exposer. En effet il ne s'agit plus ici uniquement de déplacer des brèches dans l'alignement, il faut également être en mesure de les couper au mieux. Pour cela il est nécessaire de tester tous les fractionnements possibles, opérations avec une complexité importante.

```
R----DKA---DFGAINKLE          R-------DKADFGAINKLE
R----AKE---DHSAILLNE          R-------AKEDHSAILLNE
HGVEVEVLLRHLAALPQDIQ          HGVEVEVLLRHLAALPQDIQ
E----GKS---LEEIIRSSE          E-------GKSLEEIIRSSE
   Résultat Plasma                      Référence
```

FIG. 6.3 – Exemple de brèches voisines à regrouper.

6.5.3 Implémentations réalisées dans *Plasma II*

Solutions proposées

L'utilisation d'une méthode de recherche locale semble appropriée afin de réarranger au mieux les brèches obtenues lors de la première phase d'alignement. Ce procédé permet en particulier de supprimer une partie des brèches lorsqu'une colonne complète est créée. En effet, ce cas de figure est favorisé puisque la création d'une colonne de brèche augmente le coût de l'alignement.

L'algorithme que nous avons réalisé permet de couper 'verticalement' une partie d'un alignement à la position souhaitée et de la dimension voulue. Il est possible d'appliquer au sous-alignement ainsi obtenu quelques opérations élémentaires pour déplacer les brèches :

- Déplacement d'une brèche unique appartenant à une séquence,
- Déplacement de toutes les brèches du sous-alignement qui débutent à une même position et qui ont la même longueur (cf. exemple ci-dessus).

Ces déplacements peuvent être faits sur toute la longueur du sous-alignement, la variation du coût étant donnée à chaque fois.

Une méthode possible pour réaliser un fractionnement consiste à utiliser un algorithme génétique dont chaque individu de la population est constitué d'un fractionnement différent des brèches. Le croisement entre deux individus étant réalisé en conservant les positions de brèches communes.

La version qui avait été réalisée ne permettait de réaligner que des sous alignements de 4 séquences. Les essais effectués sur un exemple codé " en dur " donnent de bons résultats. Toutefois s'était posé un problème important sur le choix du découpage de l'alignement, aucune solution n'ayant à l'époque été trouvée.

Problème rencontré

Le choix du découpage d'un sous-alignement n'est pas trivial, surtout lorsque l'alignement que l'on cherche à améliorer comporte de nombreuses brèches. De plus on ne peut pas fractionner un alignement en plusieurs sous-alignements de façon aléatoire. Lorsque l'on utilise la somme des paires (pondérée ou non), pour que la valeur de l'alignement soit égale à la somme des valeurs de chaque sous-alignement, il ne faut pas couper à proximité directe d'une brèche. En effet le coût affine utilisé pour évaluer les brèches dans un alignement va générer un surcoût lorsqu'une brèche est séparée en deux parties ou plus.

6.5.4 Implémentation

Nous avons implémenté un algorithme permettant de pallier les problèmes exposés précédemment. Pour cela nous coupons l'alignement multiple en zones mal alignées, et nous cherchons à l'améliorer. Nous avons implémenté une méthode de descente, cherchant ainsi à obtenir des améliorations successives jusqu'à ce qu'il ne soit plus possible d'améliorer le résultat. Notre fonction objectif est la somme des paires pondérée, il s'agit donc d'un problème de maximisation.

Le sous-alignement dont on cherche à améliorer la qualité représente la configuration initiale. L'ensemble des voisins correspond aux configurations que l'on obtient en réalisant une des opérations suivantes :

- Déplacement d'une unique brèche,
- Déplacement d'un ensemble de brèches superposées (comme à la figure 6.2),
- Fusion de deux brèches.

Le principe de l'algorithme est celui d'une méthode de descente : tant qu'il est possible de trouver un voisin de meilleure qualité, il devient la nouvelle solution courante. L'algorithme s'arrête lorsqu'il n'est plus possible de trouver un voisin meilleur que la solution courante.

Le nouveau sous-alignement ainsi obtenu est réinséré à sa place dans l'alignement multiple initial, et le processus est réitéré de la même façon pour les autres sous-alignements.

6.5.5 Tests et résultats

Nous avons testé l'algorithme sur les résultats obtenus par *Plasma II* à partir de jeux d'essais de *Balibase*.

Cela n'a malheureusement conduit à aucune amélioration pour l'ensemble des jeux d'essais. Sur l'ensemble des sous-alignements à réaligner, l'algorithme n'est pas parvenu à trouver un meilleur voisin.

Une des explications, selon nous, provient directement de la fonction de somme des paires. En effet lorsque l'on déplace une brèche dans une séquence, celle-ci risque de devenir moins bien alignée par rapport aux autres séquences. Il y a ainsi un fort risque de détérioration de la solution. Si l'amélioration d'une configuration nécessite le déplacement simultané de plusieurs brèches, il est fort possible que pris séparément chacun de ses déplacements représente une détérioration de la solution courante.

6.5.6 Conclusion

Dans cette partie nous avons cherché à déterminer quelles étaient les erreurs d'alignement produites par *Plasma II*. Pour cela nous avons comparé les résultats de notre algorithme présentés au chapitre précédent, avec les alignements de référence de *Balibase*. Nous avons présenté les erreurs, c'est-à-dire les différences, que l'on peut retrouver dans nos alignements.

Nous avons proposé des opérations élémentaires sur les brèches pour essayer d'améliorer notre alignement. L'objectif étant d'essayer de trouver une suite finie d'opérations permettant de passer de l'alignement obtenu par *Plasma II* à un nouvel alignement corrigé, de meilleure qualité.

Choisir une méthode de descente semble ne pas permettre d'obtenir une amélioration du résultat, signe que les opérations à effectuer pour cela doivent être réalisées simultanément. En effet, pour obtenir une meilleure solution il est nécessaire de réaliser plusieurs opérations. Or prises séparément, celles-ci détériorent la solution initiale. Nous pensons qu'une méthode permettant une redistribution plus importante des brèches pourrait permettre d'améliorer le résultat. Nous pensons en particulier à un algorithme génétique où la

population serait constituée par différentes variations des positions des brèches dans la solution courante. Pour éviter que les erreurs produites en utilisant la fonction de somme des paires pondérée ne se reproduise, il semble également intéressant de réaliser les évaluations avec une autre fonction, telle que *Coffee* ou *T-Coffee*.

Une autre direction à suivre consisterait à accepter un voisin qui n'améliore pas la solution courante, en remplaçant par exemple notre algorithme de descente par une recherche tabou.

6.6 Récapitulatif des résultats

6.6.1 Introduction

Dans cette section nous reprenons tous les meilleurs résultats obtenus dans chaque catégorie ou sous-catégorie avec les paramètres par défaut. Notre objectif est de nous rendre compte des résultats que pourrait obtenir notre algorithme s'il était en mesure de déterminer quelles options utiliser. En effet nous avons vu dans ce chapitre pour certaines catégories de jeux d'essais que les résultats peuvent être améliorés.

6.6.2 Les résultats

Nous présentons dans les tableaux 6.4 et 6.5 les résultats obtenus pour chacun des deux critères SPS et CS. Les scores obtenus par les autres algorithmes sont également rappelés pour faciliter la comparaison. *Plasma II* 1 correspond aux résultats que nous avons donnés au chapitre précédent, et *Plasma II* 2 prend en compte les améliorations obtenues dans ce chapitre.

	Ref 1	Ref 2	Ref 3	Ref 4	Ref 5	Moyenne
Clustal W	0.809	0.932	0.723	0.834	0.858	0.828
T-Coffee	0.814	0.928	0.739	0.852	0.943	0.840
MAFFT	0.829	0.931	0.812	**0.947**	**0.978**	0.867
Muscle	0.821	0.935	0.784	0.841	0.972	0.851
ProbCons	0.849	**0.943**	**0.817**	0.939	0.974	**0.880**
Plasma II 1	0.832	0.916	0.729	0.823	0.918	0.844
Plasma II 2	**0.862**	0.916	0.729	0.847	0.918	0.863

TAB. 6.4 – *Résultats avec le critère SPS.*

6.6.3 Conclusion

Cette section fait une synthèse des résultats obtenus par les différentes méthodes testées dans ce chapitre. Ainsi pour chaque référence, ou sous-catégorie de la référence 1, nous avons pris la version de l'algorithme donnant les meilleurs résultats.

Ce procédé se base sur la possibilité de déterminer à quelle catégorie appartient un jeu d'essai. En effet, en disposant d'un oracle nous indiquant quelle méthode il convient

	Ref 1	Ref 2	Ref 3	Ref 4	Ref 5	Moyenne
Clustal W	0.707	0.592	0.481	0.623	0.634	0,656
T-Coffee	0.712	0.524	0.480	0.644	0.863	0,669
MAFFT	0.736	0.525	0.595	0.822	**0.911**	0,712
Muscle	0.725	0.593	0.543	0.593	0.901	0,692
ProbCons	0.765	**0.623**	**0.631**	**0.828**	0.892	**0,747**
Plasma II 1	0.734	0,555	0.350	0.784	0.768	0.679
Plasma II 2	**0.784**	0.555	0.350	0.784	0.768	0.708

TAB. 6.5 – *Résultats avec le critère CS.*

d'employer pour aligner les séquences il est possible d'améliorer les résultats. En particulier pour la référence 1, où *Plasma II* est l'algorithme donnant les meilleurs résultats, aussi bien avec le critère SPS qu'avec le critère CS.

Déterminer si un ensemble de séquences appartient ou non à la référence 1 est assez simple. De même pour chaque sous-référence, un simple calcul de similarité permet de trouver où doit être classé le jeu de séquences.

Il semble intéressant de continuer la validation de ces méthodes. Il faudrait pour cela étendre les tests avec des jeux d'essais n'appartenant pas à *Balibase*, et vérifier si l'on obtient les mêmes améliorations.

6.7 Conclusion

Dans ce chapitre nous avons pris comme point de départ l'algorithme exposé au chapitre 5. Les résultats que nous avons obtenus étaient bons, mais nous avons toutefois cherché à modifier certaines parties de l'algorithme afin de déterminer les influences que cela peut avoir. Ces variantes ont déjà été testées par d'autres logiciels, et nous avons donc souhaité savoir si elles permettent d'améliorer ou non les résultats de notre algorithme.

Ainsi, nous avons exposé les différentes méthodes implémentées, puis nous avons cherché à valider expérimentalement si elles avaient ou non une influence sur l'alignement multiple obtenu. Les deux premières permettent d'améliorer les résultats pour certaines références, alors que les deux autres n'ont aucun effet. Comme nous pouvons le constater dans le récapitulatif des meilleurs résultats, *Plasma II* se situe alors à la première place pour la référence 1 pour les deux critères SPS et CS. Les résultats pour les autres références sont quant à eux peu ou pas améliorés.

Il nous semble donc intéressant de chercher à déterminer quelle méthode employer pour aligner un jeu de séquence. Pour cela nous pensons chercher à déterminer si un jeu d'essai appartient ou non à la référence 1. Et si tel est le cas déterminer également à quelle sous-référence il appartient.

D'une façon plus générale nous pensons qu'il est certainement intéressant de chercher à modifier également les autres paramètres d'alignements en fonction du jeu de séquences à aligner. Ainsi, nous avons pris un jeu de paramètres standard pour la matrice de substitution ainsi que pour les valeurs de K et h. Toutefois des tests groupés par références sur

d'autres paramètres semblent très intéressants. Cette approche est en cours de réalisation, mais elle n'est encore actuellement qu'à son commencement.

Conclusion Générale

Principales contributions

Cette thèse s'articule autour de deux algorithmes d'alignement multiple utilisant le même principe général des algorithmes progressifs. Ces deux algorithmes ont en commun de réaliser des alignements d'alignements. Contrairement aux algorithmes progressifs tels que *Clustal W*, nous conservons toutes les séquences au lieu d'utiliser un profil. Le but recherché étant d'améliorer la qualité de l'alignement obtenu.

Dans *Plasma*, l'alignement est réalisé par insertions successives de brèches dans l'un des deux alignements ou dans les deux. Nous cherchons pour cela à maximiser à chaque étape la fonction de somme des paires en essayant d'insérer au mieux des colonnes de brèches de dimensions différentes. L'algorithme utilisé repose sur une méthode de descente, il s'arrête donc dès qu'il arrive à un maximum local. Il détermine à chaque itération le meilleur voisin possible pour la configuration courante, en calculant toutes les possibilités pour insérer une brèche. Cet algorithme semble intéressant mais son implémentation actuelle ne permet pas de l'utiliser avec un script pour l'évaluer sur une base de jeux d'essais. Une nouvelle implémentation nous semble intéressante, avec la possibilité de choisir une autre heuristique que la descente, comme par exemple le recuit-simulé ou la recherche tabou. Il deviendra alors possible de comparer cet algorithme avec ceux que nous avons cités dans la partie de l'état de l'art, comme nous l'avons fait avec *Plasma II*.

Notre second algorithme, *Plasma II*, utilise une méthode très différente pour réaliser l'alignement de deux alignements. Il est en effet basé sur l'algorithme exact d'alignement de deux séquences. Nous proposons au travers de cet algorithme une extension de l'algorithme de *Needleman-Wunsch* pour l'alignement de deux alignements. Nous avons pour cela étendu les opérations d'édition pour k séquences, et nous avons conçu un algorithme permettant de les utiliser. Il s'agit d'un algorithme basé sur la programmation dynamique, et il nous permet d'aligner deux blocs en maximisant la fonction de somme des paires pondérée.

Nous avons testé *Plasma II* sur les jeux d'essais de *Balibase*, en utilisant les deux critères de comparaison *SPS* et *CS*. Nous avons pour cela utilisé notre algorithme selon deux protocoles différents : une version standard et une version que nous avons qualifiée de "maximale".

La version standard utilise un unique jeu de paramètres pour réaliser les alignements. Il s'agit des paramètres utilisés par défaut, à savoir $K = -10$, $h = -1$ et la matrice *Gonnet 250*. Les résultats sont bons, en effet les moyennes obtenues avec les deux critères *SPS* et *CS* sont supérieures à celles qu'obtient un algorithme tel que *Clustal W*. *Clustal W* de par sa nature est l'algorithme le plus proche de *Plasma II* puisqu'il est basé sur le principe d'alignement de profils que nous souhaitions ne pas utiliser. En revanche, nous n'obtenons

pas des résultats aussi bons que ceux des algorithmes actuellement les plus performants : *Muscle*, *MAFFT* et *ProbCons*.

Dans la version dite maximale, nous avons fait varier les trois paramètres d'alignements de notre algorithme. Pour tous les alignements ainsi générés pour chaque jeu d'essais, nous avons conservé le meilleur résultat. L'objectif que nous poursuivions consistait à déterminer l'influence que peuvent avoir ces paramètres sur les résultats donnés par notre algorithme. Nous constatons une amélioration de la qualité des résultats, en effet nous obtenons une augmentation de la moyenne pour les deux critères de comparaison. Pour le critère *SPS* cette augmentation est égale à $7,3\%$, pour le critère *CS* elle est égale à $16,9\%$. Ces valeurs permettent d'obtenir une moyenne supérieure à celles des autres algorithmes. Toutefois *ProbCons* obtient toujours les meilleurs résultats sur certaines des références de *Balibase*. Ce constat est très encourageant pour chercher à améliorer les performances de notre algorithme. Et adapter les paramètres d'alignement en fonction des séquences à aligner semble une approche intéressante pour améliorer les résultats.

Les temps de calcul sont dans la moyenne des meilleurs algorithmes actuels, et supérieurs à ceux de *Clustal W*. Ils restent toutefois raisonnables, puisqu'ils sont de l'ordre de la seconde pour un alignement de 24 séquences. Les temps d'exécutions pour les algorithmes auxquels nous comparons *Plasma II* sont suffisamment faibles pour ne pas être pris comme un critère déterminant d'efficacité.

Afin d'augmenter encore la qualité des résultats de notre algrithme nous avons choisi d'ajouter deux autres méthodes de calculs (brèches de début et de fin à coûts réduits et *Coffee*) pour l'évaluation faite lors de la phase de calculs. Ces méthodes ont une influence significative sur le résultat final puisqu'en utilisant celle qui convient le mieux pour chaque catégorie, les résultats peuvent être améliorés. Ainsi obtient-on le meilleur résultat pour la référence 1 de *Balibase*, devant les algorithmes actuellement les plus compétitifs.

Nous avons également testé deux autres méthodes qui n'ont pas amélioré nos résultats. La première consiste à construire un nouveau *guide-tree* à partir de l'alignement multiple qui a été calculé, puis à l'utiliser pour réaligner les séquences. Nous n'avons malheureusement obtenu aucune amélioration. La seconde méthode avait pour objectif de réaliser une post-optimisation à la fin de l'alignement. Pour cela nous avons essayé de détecter des erreurs d'alignement, afin de les corriger. Là encore la méthode utilisée ne nous a pas permis d'améliorer la qualité de nos résultats.

Perspectives de recherches

Dans un futur proche nous envisageons d'effectuer des tests plus poussés, en cherchant en particulier à utiliser d'autres jeux de paramètres, puisque comme nous l'avons vu ils influent grandement sur la qualité du résultat. L'objectif est de pouvoir déterminer pour chaque jeu d'essais des paramètres et une fonction d'évaluation adaptés pour réaliser l'alignement multiple.

Jusqu'à présent nous avons testé *Plasma II* sur les jeux de *Balibase*, car ce sont les plus souvent pris comme référence. D'autre bases de jeux d'essai existent également, comme

par exemple *Homstrad*, *Oxbench* ou *Prefab*, et pour valider complètement les résultats, il semble très intéressant d'obtenir également les résultats avec ces jeux d'essais.

Nous pensons également que la post-optimisation du résultat final est une piste prometteuse. Cette méthode pouvant bien entendu être utilisée pour *Plasma II* mais également pour essayer d'améliorer le résultat de tout autre algorithme. Elle semble également être intéressante pour une nouvelle implémentation du premier algorithme *Plasma*.

Si le résultat obtenu comporte des erreurs, celles-ci proviennent probablement d'erreurs qui se sont répercutées lors des alignements successifs. Une correction sur un nombre plus faible de séquences semble être plus aisée et probablement plus efficace. Nous pensons donc qu'une post-optimisation après chaque alignement d'alignements devrait améliorer la qualité du résultat final. Cette méthode peut là encore être utilisée aussi bien pour *Plasma* que pour *Plasma II*.

La fonction d'évaluation de *T-Coffee* permet d'obtenir de meilleurs résultats que celle de *Coffee*. Il semble donc intéressant de l'implémenter afin de comparer la qualité des alignements obtenus. De plus, si l'alignement d'alignements est effectué en utilisant la fonction de somme de paires pondérée, réaliser une post-optimisation avec la même fonction n'est pas nécessairement intéressant. Il semble judicieux d'utiliser une fonction différente, comme celle de *T-Coffee* pour cette opération. En effet, si l'utilisation de la somme des paires pondérée engendre une erreur, il se peut qu'elle ne soit pas en mesure de les corriger.

Références bibliographiques

[Aart and van Laarhoven, 1987] cité page 90
E.H.L. Aart and P.J.M. van Laarhoven. *Simulated Annealing : a Review of Theory and Applications*. Kluwer Academic Publishers, 1987.

[Altschul *et al.*, 1989] cité page 51
S.F. Altschul, R.J. Carroll, and D.J. Lipman. Weights for data related by a tree. *J. Molec. Biol.*, 208 :647–653, 1989.

[Altschul *et al.*, 1990] cité page 47
S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, Vol. 215(3) :403–410, 1990.

[Altschul, 1989a] cité page 29
S.F. Altschul. Gap costs for multiple sequence alignment. *J. Theor. Biol.*, 138 :297–309, 1989.

[Altschul, 1989b] cité page 104
S.F. Altschul. Gap costs for multiple sequence alignment. *Journal of Theoretical Biology*, 138 :297–309, 1989.

[Bairoch, 1991] cité page 12
A. Bairoch. Prosite : a dictionnary of sites and patterns in proteins. *Nucleic Acids Research*, 19 :2241–2245, 1991.

[Baxevanis, 2000] cité page 11
A.D. Baxevanis. The molecular biology database collection : an online compilation of relevant database resources. *Nucleic Acid Research*, 28, 2000.

[Bernstein *et al.*, 1977] cité page 11
F.C. Bernstein, T.F. Koetzle, G.J. Williams, E.F. Meyer Jr, M.D. Brice, J.R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The protein data bank : a computer-based archival file for macromolecular structures. *J. Mol. Biol.*, 112 :535–542, 1977.

[Carillo and Lipman, 1988] cité page 59
H. Carillo and D.J. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48 :1073–1082, 1988.

[Chenna *et al.*, 2003] cité page 62
R. Chenna, H. Sugawara, T. Koike, R. Lopez, T.J. Gibson, D.G. Higgins, and J.D. Thompson. Multiple sequence alignment with the clustal series of programs. *Nucleic Acids Research*, 31(13) :3497–3500, 2003.

[Chou and Fasman, 1974] cité page 7, 13
P.Y. Chou and G.D. Fasman. Prediction of protein conformation. *Biochemistery*, 13 :222–245, 1974.

[Clote and Backofen, 2000] cité page 23
P. Clote and R. Backofen. *Computational Molecular Biology - An Introduction*. Mathematical and Computational Biology. Wiley, 2000.

[Cohen *et al.*, 1994] cité page 29
M.A. Cohen, S.A. Benner, and G.H. Gonnet. Analysis of mutation during divergent evolution : The 400 by 400 dipeptide mutation matrix. *Biochem. Biophys. Res. Commun.*, 199 :489–496, 1994.

[Cook, 1971] cité page 16
S. Cook. The complexity of theorem proving procedures. In *Proceedings of the third annual ACM symposium on Theory of Computing*, pages 151–158, 1971.

[Cori and Lascar, 2003] cité page 13
R. Cori and D. Lascar. *Logique mathématique*. Dunod, 2003.

[Corpet, 1988] cité page 66, 121
F. Corpet. Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Research*, Vol. 22 :10881–10890, 1988.

[Dayhoff *et al.*, 1978] cité page 26
M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model for evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, Vol. 5, 1978.

[Derrien *et al.*, 2002] cité page 81, 88
V. Derrien, J-M. Richer, and J-K. Hao. Plasma : A new hybrid approach to multiple sequence alignment. In *Proceedings of PPSN VII Workshop on Advances in Nature-Inspired Computation*, pages 31–32, 2002.

[Derrien *et al.*, 2003a] cité page 81
V. Derrien, J-M. Richer, and J-K. Hao. Plasma : un algorithme hybride pour le problème d'alignement multiple de séquence. *Journées Nationales sur la résolution Pratique de Problèmes NP-Complets (JNPC'03)*, pages 345–348, 2003.

[Derrien *et al.*, 2003b] cité page 81
V. Derrien, J-M. Richer, and J-K. Hao. Plasma, une approche hybride pour l'alignement multiple de séquences. *Actes des 5ème Congrès National sur la Recherche Opérationnelle et l'Aide à la Décision (ROADEF-03)*, pages 77–78, 2003.

[Derrien *et al.*, 2005] cité page 91
V. Derrien, J-M. Richer, and J-K. Hao. Plasma, un nouvel algorithme progressif pour l'alignement multiple de séquences. *Journées Francophones de Programmation par Contraintes (JFPC¿05)*, pages 39–48, 2005.

[Do *et al.*, 2005] cité page 68
C.B. Do, M.S.P. Mahabhashyam, M. Brudno, and S. Batzoglu. Probcons : Probabilistic consistency-based multiple sequence alignment. *Genome Res.*, 34 :227–230, 2005.

[Eddy, 1995] cité page 71
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)*, pages 114–120, 1995.

[Edgar, 2004a] cité page 67
R.C. Edgar. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Research*, Vol. 32 :380–385, 2004.

[Edgar, 2004b] cité page 67, 76
R.C. Edgar. Muscle : multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, Vol. 32 :1792–1797, 2004.

[Feng and Doolittle, 1990] cité page 62
D.F. Feng and R.F Doolittle. Progressive alignment and phylogenetic tree construction of protein sequences. *Methods in Enzymology*, 183 :375–387, 1990.

[Fitch and Margoliash, 1967] cité page 7, 12
W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155 :279–284, 1967.

[Garey and Johnson, 1979] cité page 13
M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completness*. W. H. Freeman, 1979.

[Gibas and Jambeck, 2002] cité page 12
C. Gibas and P. Jambeck. *Introduction à la bioinformatique*. O'Reilly, 2002.

[Giegerich and Wheeler, 1996] cité page 20
R. Giegerich and D. Wheeler. Pairwise sequence alignment, 1996.

[Glover and Laguna, 1997] cité page 71
F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.

[Gotoh, 1982] cité page 40
O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, Vol. 162 :705–708, 1982.

[Hamming, 1950] cité page 22
R.W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2) :147–160, 1950.

[Harry, 2001] cité page 8
M. Harry. *Génétique moléculaire et évolutive*. Maloine, 2001.

[Haussler *et al.*, 1993] cité page 71
D. Haussler, A. Krogh, I.S. Mian, and K. Sjölander. Protein modeling using hidden markov models : Analysis of globins. In *Proceedings of the Hawaii International Conference on System Sciences*, 1993.

[Henikoff and Henikoff, 1991] cité page 27
S. Henikoff and J.G. Henikoff. Automated assembly of protein blocks for database searching. *Nucl. Acids Res.*, 19 :6565–6572, 1991.

[Henikoff and Henikoff, 1992] cité page 27
S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. In *Proceedings of the National Academy of Science*, volume Vol. 89, pages 10915–10919, 1992.

[Hogeweg and Hesper, 1984] cité page 61
P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phylogenetic trees. an integrated method. *J. Mol. Evol.*, 20 :175–186, 1984.

[Holland, 1975] cité page 70
J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Hulo *et al.*, 2006] cité page 13
N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P.S. Langendijk-Genevaux, M. Pagni, and C.J.A. Sigrist. The prosite database. *Nucleic Acids Research*, 15 :330–340, 2006.

[Karplus and Hu, 2001] cité page 73
K. Karplus and B. Hu. Evaluation of protein multiple alignments by sam-t99 using the balibase multiple alignment test set. *Bioinformatics*, 17(8) :713–720, 2001.

[Katoh *et al.*, 2002] cité page 68
K. Katoh, K. Misawa, K. Kuma, and T. Miyata. Mafft : a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Research*, Vol. 30 :3059–3066, 2002.

[Katoh *et al.*, 2005] cité page 106
K. Katoh, K. Misawa, K. Kuma, and T. Miyata. Mafft version 5 : improvement in accuracy of multiple sequence alignement. *Nucleic Acids Research*, Vol. 33 :511–518, 2005.

[Kececioglu and Starrett, 2004] cité page 2, 3
J.D. Kececioglu and D. Starrett. Aligning alignments exactly. In *Proceedings of the 8th ACM Conference on Research in Computational Molecular Biology*, pages 85–96, 2004.

[Kececioglu and Zhang, 1998] cité page 2, 92, 104, 104, 104
J.D. Kececioglu and W. Zhang. Aligning alignments. In Springer-Verlag Lecture Notes in Computer Science, editor, *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, volume 1448, pages 189–208, 1998.

[Kececioglu *et al.*, 2000] cité page 71
J.D. Kececioglu, H-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, and M. Vingron. A polyhedral approach to sequence alignment problems. *Discrete Appl. Math.*, 104(1-3) :143–186, 2000.

[Kim *et al.*, 1994] J. Kim, S. Pramanik, and M.J. Chung. Multiple sequence alignment using simulated annealing. *Bioinformatics*, Vol. 10 :419–426, 1994.

[Levenshtein, 1966] cité page 24
V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10 :707–710, 1966.

[Ma *et al.*, 2003] cité page 104, 104, 104
Bin Ma, Zhuozhi Wang, and Kaizhong Zhang. Alignment between two multiple alignments. *CPM'03*, pages 254–265, 2003.

[Mizuguchi *et al.*, 1998] cité page 76
K. Mizuguchi, C.M. Deane, T.L. Blundell, and J.P. Overington. Homstrad : A database of protein structure alignments for homologous families. *Protein Science*, Vol. 7 :2469–2471, 1998.

[Morgenstern, 1999] cité page 54, 69
B. Morgenstern. Dialign2 : improvment of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, Vol. 15 (3) :211–218, 1999.

[Needleman and Wunsch, 1970] cité page 7, 12, 35
S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3) :443–453, 1970.

[Notredame and Higgins, 1996] cité page 70
C. Notredame and D.G. Higgins. Saga : Sequence alignment by genetic algorithm. *Nucleic Acid Research*, Vol. 24 :1515–1524, 1996.

[Notredame *et al.*, 1998] cité page 52, 116
C. Notredame, L. Holme, and D.G. Higgins. Coffee : A new objective function for multiple sequence alignmnent. *Bioinformatics*, Vol. 14 (5) :407–422, 1998.

[Notredame *et al.*, 2000] cité page 53
C. Notredame, D.G. Higgins, and J. Heringa. T-coffee : A novel method for multiple sequence alignments. *Journal of Molecular Biology*, Vol. 302 :205–217, 2000.

[Notredame, 2002] cité page 12, 55
C. Notredame. Recent progresses in multiple sequence alignment : a survey. *Pharmacogenomics*, 3, 2002.

[O'Sullivan *et al.*, 2004] cité page 53
O. O'Sullivan, K. Suhre, C. Abergel, D.G. Higgins, and C. Notredame. 3dcoffee : Combining protein sequences and structures within multiple sequence alignments. *Journal of Molecular Biology*, Vol. 340 :385–395, 2004.

[Pearson and Lipman, 1988] cité page 11, 48
W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. In *Proc Natl Acad Sci U S A*, volume 85(8), pages 2444–2448, 1988.

[Raghava *et al.*, 2003] cité page 76
G.P.S. Raghava, S.M.J Searle, P.C. Audley, J.D. Barber, and G.J. Barton. Oxbench : A benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, Vol. 4 :47, 2003.

[Riaz *et al.*, 2004] cité page 71
T. Riaz, Y. Wang, and K.B. Li. Multiple sequence alignment using tabu search. In *CRPIT '29 : Proceedings of the second conference on Asia-Pacific bioinformatics*, pages 223–232. Australian Computer Society, Inc., 2004.

[Richer *et al.*, 2007] cité page 91
JM. Richer, V. Derrien, and JK. Hao. A new dynamic programming algorithm for multiple sequence alignment. *Proceedings of the First International Conference on Combinatorial Optimization and Applications (COCOA 07). Lecture Notes in Computer Science 4616*, pages 52–61, 2007.

[Saitou and Nei, 1987] cité page 51, 64
N. Saitou and M. Nei. The neighbor-joining method : a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4 :406–425, 1987.

[Shi *et al.*, 2001] cité page 54
J. Shi, T.L. Blundell, and K. Mizuguchi. Fugue : sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *J. Mol. Biol.*, 310 :243–257, 2001.

[Smith and Waterman, 1981] cité page 45
T.M. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, Vol. 147 :195–197, 1981.

[Sokal and Michener, 1958] cité page 64
R.R. Sokal and C.D. Michener. A statistical method for evaluating systemaic relationships. *University of Kansas Science Bulletin*, Vol. 38 :1409–1438, 1958.

[Stebbings and Mizuguchi, 2004] cité page 76
L.A. Stebbings and K. Mizuguchi. Homstrad : recent developments of the homologous protein structure alignment database. *Nucleic Acid Research*, Vol. 32 :203–207, 2004.

[Stoye *et al.*, 1997] cité page 60
J. Stoye, V. Moulton, and A.W. Dress. Dca : an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. *Comput. Apll. Biosci.*, 13 (6) :625–626, 1997.

[Subramanian *et al.*, 2005] cité page 69
A.R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. Dialign-t : An improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6 :66, 2005.

[Thompson *et al.*, 1994] cité page 62, 65
J.D. Thompson, D.G. Higgins, and 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 :4673–4680, 1994.

[Thompson *et al.*, 1999] cité page 73
J.D. Thompson, F. Plewniak, and O. Poch. Balibase : A benchmark alignments database for the evaluation of multiple sequence alignment programs. *Bioinformatics*, Vol. 15 :87–88, 1999.

[Thompson *et al.*, 2001] cité page 54
J.D. Thompson, F. Plewniak, R. Ripp, J-C. Thierry, and O. Poch. Towards a reliable objective function for multiple sequence alignments. *Journal of Molecular Biology*, Vol. 314 :937–951, 2001.

[Thompson *et al.*, 2005] cité page 74
J.D. Thompson, P. Koehl, R. Ripp, and O. Poch. Balibase 3.0 : Latest developments of the multiple sequence alignment benchmark. *PROTEINS : Structure, Function, and Bioinformatics*, 61 :127–136, 2005.

[Turing, 1950] cité page 13
A.M. Turing. Computing machinery and intelligence. *Mind*, LIX (236) :433–460, 1950.

[Turing, 1995] cité page 13
A.M. Turing. *La machine de Turing*. Points Sciences. Ed. du Seuil, 1995.

[Vingron and Argos, 1989] cité page 51
M. Vingron and P. Argos. A fast and sensitive multiple sequence alignment algorithm. *Comput. Appl. Biosci*, 5 :115–121, 1989.

[Wallace *et al.*, 2005] cité page 12
I.M. Wallace, G. Blackshields, and D.G. Higgins. Multiple sequence alignments. *Current Opinion in Structural Biology*, 15 :261–266, 2005.

[Wallace *et al.*, 2006] cité page 53
I.M. Wallace, O. O'Sullivan, D.G. Higgins, and C. Notredame. M-coffee : combining multiple sequence alignment methods with t-coffee. *Nucleic Acids Research*, Vol. 34 :1692–1699, 2006.

[Walle *et al.*, 2005] cité page 76
I. Van Walle, I. Lasters, and L. Wyns. Sabmark - a benchmark for sequence alignment that cover the entire known fold space. *Bioinformatics*, Vol. 21(7) :1267–1268, 2005.

[Wang and Jiang, 1994] cité page 55
L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4) :337–348, 1994.

[Watson and Crick, 1953] cité page 7
James Watson and Francis Crick. Molecular structure of nucleic acids. *Nature*, 4356 :737–738, 1953.

[Wheeler and Kececioglu, 2007] cité page 3, 104
Travis J. Wheeler and John D. Kececioglu. Multiple alignmen by aligning alignments. *ISMB/ECCB 2007*, vol. 23 :559–568, 2007.

[Wilf, 1989] cité page 13
H.S. Wilf. *Algorithmes et complexité*. Logique Mathématiques Informatique. Masson, 1989.

Heuristiques pour la résolution du problème d'alignement multiple

Résumé

L'alignement multiple est une opération permettant de mettre en évidence la similarité entre plusieurs séquences. Il est notamment utilisé pour la reconstruction de phylogénies, la recherche de motifs et la prédiction de structures. Cette thèse s'intéresse au développement de nouveaux algorithmes pour ce problème particulièrement difficile, et introduit deux algorithmes progressifs ayant pour point commun de réaliser un alignement multiple par alignements successifs de groupes de séquences.
Le premier algorithme, *Plasma* utilise une méthode de descente, dont chaque itération consiste à réaliser des insertions de colonnes de brèches dans deux alignements multiples à aligner. Le second algorithme, *Plasma II*, est basé sur le principe de la programmation dynamique. Nous généralisons ici l'algorithme utilisé pour l'alignement de deux séquences, et étendons le cadre de la programmation dynamique à l'alignement de deux alignements multiples. Cet algorithme ainsi que plusieurs variantes sont intensivement évalués sur les jeux d'essais de *Balibase*, montrant des résultats encourageants, voire compétitifs, par rapport à certains algorithmes de référence comme *Clustal W*, tant sur la qualité de l'alignement que sur le temps de calcul.

Mots-clés : Bioinformatique, Alignement Multiple de Séquences, Programmation Dynamique.

Heuristics for the multiple alignment problem

Abstract

Multiple alignment is one of the basic and central tasks in *Bioinformatics* which tries to higlight similarities between sequences. It is a prior to phylogeny reconstruction, pattern matching and protein structure prediction. This thesis aims to develop new algorithms to tackle this problem, and it introduces two new progressive algorithms that align alignments instead of profiles.
The first algorithm, *Plasma I*, uses a descent where each iteration consists in gap insertions in one of the two multiple alignments. The second algorithm, *Plasma II* is based on the Dynamic Programming principle. We propose with this algorithm a generalization of the pairwise sequence alignment algorithm, and we extend the Dynamic Programming framework to align two multiple alignments. The performances of *Plasma II* are assessed on the well-known *Balibase* benchmarks, and compared with several algorithms. *Plasma II* provides results of quality with fast computation time.

Keywords : Bioinformatics, Multiple Sequence Alignment, Dynamic Programming.