Stratégies d'optimisation combinatoire pour le problème de l'alignement local multiple sans indels, et application aux séquences protéiques

Hernandez, David

UNIVERSITÉ DE GENÈVE — FACULTÉ DES SCIENCES

Département d'informatique — Professeur R.D. Appel
Institut Suisse de Bioinformatique — Docteur R. Gras

Stratégies d'optimisation combinatoire pour le problème de l'alignement local multiple sans indels, et application aux séquences protéiques

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention bioinformatique

par

David Hernandez

de

Versoix (GE)

Thèse N° 3640

GENÈVE
2005

La Faculté des sciences, sur le préavis de Messieurs R. D. APPEL, professeur associé et directeur de thèse (Département d'informatique), R. GRAS, docteur et co-directeur de thèse (Institut Suisse de bioinformatique - Genève, Suisse), B. CHOPARD, professeur adjoint (Département d'informatique), A. DANCHIN, docteur (Institut Pasteur - Génétique des génomes bactériens - Paris, France) et E. TALBI, professeur (Ecole polytechnique de Lille - Laboratoire d'informatique fondamentale - Villeneuve D'Ascq, France) autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 29 juin, 2005

Thèse - 3640 -

Le Doyen, Pierre Spierer

Résumé

L'alignement local multiple et sans indels est une procédure classique en bioinfomatique. Elle consiste à déterminer à partir d'un ensemble de séquences supposées apparentées les n facteurs non recouvrants de taille W qui sont maximalement conservés. La mesure de conservation classiquement utilisée est l'entropie relative. Quatre modèles de contraintes concernant la répartition autorisée des facteurs choisis sont considérés : 1) exactement un facteur par séquence, 2) au plus un facteur par séquence, 3) au moins un facteur par séquence, 4) zéro, un ou plusieurs facteurs par séquence. Dans la littérature, le problème est principalement abordé d'un point de vue statistique. Nous proposons de l'approcher sous la forme d'une optimisation combinatoire par voisinage. L'optimisation est effectuée par un grimpeur strict sur deux fonctions de voisinage spécifiques. Nous montrons que cette approche plus simple est souvent préférable en terme de temps CPU ainsi qu'en terme de nombre d'évaluations de la fonction objectif. Nous proposons également une nouvelle fonction objectif : l'entropie recouvrante, laquelle est dédiée à l'alignement de séquences protéiques. Cette fonction permet, contrairement à la fonction classique, de prendre en compte le fait que certaines substitutions d'acides aminés sont plus probables que d'autres. L'entropie recouvrante présente des avantages significatifs, aussi bien du point de vue de la reconnaissance du signal (pertinence biologique), que de son effet favorable sur le paysage d'exploration, rendant l'optimisation par le grimpeur strict significativement plus efficace. Une implémentation appelée Nomad (Neighborhood Optimization for Multiple Alignment Discovery) est disponible sur www.expasy.org/tools/nomad.html.

Remerciements

Je tiens à remercier Ron Appel pour m'avoir accueilli dans le Proteome Informatics Group durant ces cinq années de thèse. Sa bienveillance ainsi que la confiance qu'il m'a accordée ont été précieuses.

Je remercie Robin Gras pour avoir encadré mon travail. Ses conseils, son écoute ainsi que son investissement ont été un atout majeur. Merci Robin, ces années furent un véritable plaisir.

Ce travail n'aurait pas été possible si je n'avais pu compter sur Nadia, ma chère épouse, qui a assuré les mille choses indispensables au fonctionnement de notre famille et au bien être de nos deux filles Liv et Loïsia. Merci infiniment Nadia !

Merci à Frédérique Lisacek, pour ses nombreux conseils scientifiques et pour l'intérêt constant qu'elle a porté à l'avancement de mes travaux.

Merci à Jacques Nicolas, pour l'intérêt qu'il a porté à mes travaux et pour ses invitations à l'IRISA de Rennes dans son groupe "SYMBIOSE".

Merci à Markus Müller, pour sa disponibilité, pour ses explications et conseils scientifiques.

Merci à Andrea Auchincloss, Patricia Hernandez et Nadine Zangger, pour leur travaux de relecture et correction de mes textes.

Merci à Christine Hoogland, qui s'est occupée de l'interface WEB sur ExPASy de Nomad, l'outil d'alignement résultant de cette thèse.

Table des matières

Remerciements **iii**

Tables des Figures **vii**

1 Introduction et motivations **3**
1.1 Notions de biologie moléculaire 4
1.1.1 L'ADN 4
1.1.2 L'ARN 5
1.1.3 La réplication de l'ADN 5
1.1.4 Les protéines 5
1.1.5 La synthèse des protéines 6
1.1.6 Les mutations et l'évolution 8
1.2 Motivations 11
1.3 Organisation du document 12

2 Optimisation combinatoire par voisinage **15**
2.1 Introduction 15
2.2 Les métaheuristiques 16
2.3 L'espace de recherche 17
2.4 Un modèle de paysage 17
2.4.1 Un modèle de paysage 18
2.4.2 Les maximums locaux 18
2.4.3 Les bassins d'attraction 20
2.4.4 La complexité des problèmes 20
2.5 Les grimpeurs 21
2.5.1 Les grimpeurs stricts 21
2.5.2 La recherche taboue 21
2.5.3 Les grimpeurs probabilistes 22
2.5.4 Le recuit simulé 22
2.6 Les algorithmes évolutionnistes 23
2.6.1 Introduction 23
2.6.2 L'algorithme génétique simple 24
2.6.3 Les approches évolutionnistes par découverte de dépendances 26
2.7 Discussion 27

3 Etat de l'art de la découverte de similarités 29
3.1 Introduction . . . 29
3.2 Découverte de patterns . . . 30
3.2.1 Généralités . . . 30
3.2.2 Recherche exhaustive . . . 32
3.2.3 Recherche par profondeur dans un arbre . . . 32
3.2.4 Recherche de patterns guidée par l'ensemble de séquences . . . 33
3.2.5 Recherche de patterns présentant des occurrences avec erreurs . . . 34
3.3 Alignements deux à deux de séquences . . . 36
3.3.1 Le système de score pour l'alignement de protéines . . . 37
3.3.2 L'algorithme de Needleman&Wunch . . . 40
3.4 Alignements multiples de séquences . . . 41
3.4.1 Généralisation de la programmation dynamique à des espaces multidimensionnels . . . 42
3.4.2 L'alignement multiple progressif . . . 42
3.4.3 ClustalW . . . 43
3.4.4 Alignements multiples utilisant un score basé sur la consistance . . . 44
3.5 Alignements locaux multiples sans indels . . . 45
3.5.1 Le Gibbs Site Sampler . . . 48
3.5.2 MEME . . . 49
3.5.3 Le Gibbs Motif Sampler et dérivés . . . 51
3.5.4 Consensus . . . 52

4 Une approche pour l'alignement local multiple 53
4.1 Introduction . . . 53
4.2 L'alignement local multiple sans indels . . . 53
4.2.1 Introduction . . . 53
4.2.2 Notations . . . 54
4.3 Entropie et contenu en information . . . 55
4.4 L'entropie relative comme fonction objectif . . . 56
4.5 L'entropie relative recouvrante . . . 59
4.5.1 Une mesure de recouvrement . . . 61
4.5.2 Intégration des valeurs de recouvrement à l'entropie relative . . . 62
4.5.3 Représentation d'un alignement . . . 66
4.6 Stratégies d'optimisation . . . 66
4.6.1 Quatre modèles de contrainte pour la répartition des occurrences . . . 67
4.6.2 Première stratégie : un grimpeur strict sur des graines aléatoires . . . 72
4.6.3 Deux stratégies supplémentaires pour le mode OOPS . . . 74
4.6.4 Résumé des trois stratégies et discussion . . . 80

5 Résulats 83
5.1 introduction . . . 83
5.2 Le problème challenge . . . 84
5.3 Un ensemble de protéines HTH . . . 85
5.3.1 Le motif HTH . . . 85
5.3.2 Comparaisons des fonctions sur les données HTH . . . 87
5.3.3 Comparaisons d'optimisation avec les méthodes classiques . . . 89
5.4 Données artificielles . . . 91

5.4.1 Le générateur Rose 91
5.4.2 Méthode 94
5.4.3 Comparaison des fonctions objectif 95
5.5 Un ensemble de protéines EF-hand 97
5.5.1 Le motif EF-hand 97
5.5.2 Comparaison des fonctions objectif 100
5.5.3 Comparaison avec les méthodes classiques 100

6 Conclusions et perspectives **103**
6.1 Conclusions 103
6.2 Perspectives 104

Bibliographie **106**

Table des figures

1.1 Structure de l'ADN 5
1.2 Réplication de l'ADN 6
1.3 Synthèse d'une protéine 9

2.1 Topologie d'un paysage de Hamming 19
2.2 Topologie d'un paysage de recombinaison 19
2.3 Opérateur de recombinaison à un point 26
2.4 BOA : réseau bayésien 28

3.1 Matrice de substitution BLOSUM62 39
3.2 L'algorithme de Needlaman & Wunsch 41
3.3 L'alignement multiple par construction 43
3.4 Matrices position spécifiques 47

4.1 L'alignement local multiple sans indels 54
4.2 Matrice de fréquences relatives 57
4.3 Matrice de recouvrements 63
4.4 Comportement de l'entropie relative recouvrante 64
4.5 Représentation d'un alignement 66
4.6 Quatre façons de contraindre la répartition des occurrences 68
4.7 Représentation des deux voisinages 70
4.8 L'opérateur de projection $MtoP$ 77
4.9 L'opérateur de projection $PtoM$ 78
4.10 Opérateurs génétiques sur les mots 79
4.11 La stratégie AG 81

5.1 Alignement des domaines HTH 87
5.2 Sensibilité au paramètre W sur les données HTH 88
5.3 Performance des fonctions objectifs sur les données HTH 89
5.4 Comparaison de quatre stratégies d'optimisation 91
5.5 Rose : un arbre évolutionnaire 93
5.6 Graphiques de comparaison sur la détection de signaux 97
5.7 Graphiques de comparaison sur les capacités d'optimisation 98
5.8 Alignement des domaines EF-hand 99
5.9 Comparaison des fonctions sur les données EF40 100

Chapitre 1

Introduction et motivations

L'analyse informatique des données issues de la biologie, et en particulier de la biologie moléculaire a donné naissance à une nouvelle discipline : la bioinformatique. Cette discipline a connu un essor considérable depuis une quinzaine d'années car de grandes quantités de données, en particulier de séquences d'ADN et de protéines, ont été rendues disponibles grâce à des techniques de laboratoire, rendant le processus de séquençage de masse possible. Le séquençage est le procédé qui consiste à déchiffrer l'enchaînement des composés de base qui constituent la séquence de ces macromolécules linéaires. Plus qu'un simple service à l'intention des biologistes, la bioinformatique a ouvert de nouveaux domaines de recherche, qui utilisent des concepts issus aussi bien de l'informatique, des statistiques que des mathématiques.

Le travail présenté dans cette thèse porte sur le problème de l'alignement multiple de séquences de biopolymères, et en particulier des séquences de protéines. Cette procédure, centrale en bioinformatique, consiste à comparer deux séquences ou un ensemble de séquences afin de révéler des *similarités* entre elles. Lorsque ces similarités sont suffisamment fortes, ou suffisamment significatives, on peut alors supposer leur *homologie*, complète ou de certaines régions de leur séquence. Le terme d'homologie a une signification précise, qui indique que ces séquences ont été produites à partir d'une même séquence ancestrale, par transformations successives au cours de l'évolution. Bien que l'homologie n'est jamais directement établie, une similarité significative au niveau de la séquence la rend très probable.

Le type de données que nous traitons se présente sous la forme d'un texte, écrit sur un cardinal de taille quatre pour l'ADN, et de taille vingt pour les protéines. Il est certes réducteur de résumer ces molécules à un simple texte, ne permettant pas de modéliser leur structure, leur conformation spatiale et leurs propriétés physico-chimiques. Il apparaît pourtant que la séquence, c'est à dire l'ordre d'enchaînement des composés de base, contient une grande partie de l'information permettant la caractérisation de ces molécules. L'analyse et la comparaison de ces séquences permet de révéler beaucoup d'éléments sur leurs fonctionnalités, ainsi que sur leur histoire évolutive.

Nous allons poursuivre cette introduction par une présentation rapide de la nature de ces macromolécules linéaires ainsi que des mécanismes de base qui leur sont associés. Nous décrirons ensuite les motivations qui sont à la base de ce travail, puis nous finirons par une description des différents chapitres de ce document.

1.1 Notions de biologie moléculaire

Cette section est destinée à donner un aperçu des concepts biologiques liés à ce travail. Le lecteur non-biologiste qui souhaite approfondir ses connaissances pourra trouver des descriptions détaillées dans (Alberts et al., 2004), un ouvrage de référence en biologie moléculaire, et d'une approche relativement facile.

1.1.1 L'ADN

En 1928, Fred Griffith montra à l'aide d'une expérience impliquant deux souches de la bactérie Pneumococcus, qu'une caractéristique pouvait être transmise d'une souche à l'autre (Griffith, 1928). Le vecteur permettant la transmission de cette caractéristique était alors inconnu. C'est en 1944 que ce vecteur a pu être purifié et identifié comme de l'acide désoxyribonucléique (ADN) (Avery et al., 1944), mais sa structure était toujours inconnue. Il fallut attendre les travaux de cristallographie de Rosalind Franklin, et la publication de James Watson et Francis Crick pour connaître la fameuse structure en double hélice (Watson and Crick, 1953).

L'information héréditaire de tous les organismes vivants, à l'exception de certains virus, est contenue dans une ou des molécules d'ADN. Cette molécule est composée de deux brins linéaires, enroulés l'un autour de l'autre, pour former une double hélice. Les fonctions de cette molécule ne se limitent cependant pas à un stockage d'information. L'ADN de par sa structure ainsi que de par ses propriétés physico-chimiques participe activement au processus de transmission et d'utilisation de cette information.
L'ADN est constitué de l'appariement de deux brins, chacun consistant en un assemblage linéaire de *nucléotides* ou *bases*. Les nucléotides utilisés sont au nombre de quatre : l'adénine 'A', la cytosine 'C', la guanine 'G', et la thymine 'T'. Ces nucléotides sont disposés sur une ossature linéaire formée de désoxiribose et de phosphate. Cette ossature est polarisée et chacune des extrémités est caractérisée par la notation $5'$ et $3'$. La double hélice est formée par l'appariement de deux brins *complémentaires*, par l'intermédiaire des nucléotides qui les composent. Le nucléotide 'G' peut s'apparier avec le 'C', et le 'A' avec le 'T'. L'appariement des deux brins est *antiparallèle*, leur polarisation étant inversée. La Figure 1.1 montre une représentation schématique de cet assemblage en double hélice antiparallèle. Étant donné que les deux brins sont complémentaires, un seul des deux est nécessaire pour représenter l'information de la séquence d'une molécule d'ADN. Pour cette raison, une séquence, ou une partie de séquence d'ADN peut être représentée sous la forme d'un texte écrit sur l'alphabet {A,T,C,G}. Par convention, le texte du brin considéré est écrit dans le sens $5' \longrightarrow 3'$.

L'ADN d'un organisme est souvent composé de plusieurs chaînes disjointes, structurées dans des *chromosomes*. Le *génome* dénomme l'ensemble de l'information portée par les chromosomes. Chaque cellule d'un organisme possède une copie ou un nombre donné de copies complètes du génome. La taille d'un génome est mesurée en nombre de paires de bases (bp). Cette taille est de l'ordre de 10^6 bp pour les bactéries, et d'environ $3.4 \cdot 10^9$ bp pour l'espèce humaine.

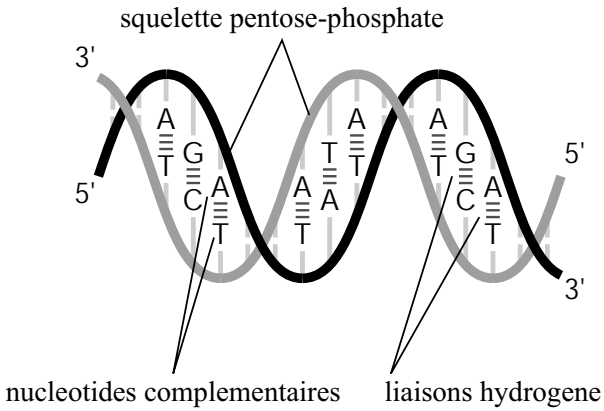


Fig. 1.1 : Une représentation schématique de la structure d'un double brin d'ADN. On peut observer l'appariement antiparallèle des brins. Les deux brins complémentaires sont appariés par le biais des nucléotides. (Illustration modifiée d'un original public et disponible à l'adresse WEB http ://roso.epfl.ch/nm/public/FIG/adn.eps)

1.1.2 L'ARN

L'ARN (acide ribonucléique) est une molécule linéaire similaire à l'ADN, formée par l'enchaînement de nucléotides. Les nucléotides d'ADN et d'ARN sont identiques, à l'exception de la thymine (T), qui est remplacée dans l'ARN par l'uracile, que l'on dénote par le symbole 'U'. Tous comme la thymine, l'uracile peut former un appariement avec l'adénine (A). L'ossature de l'ADN est formée de riboses et de phopshates. L'ARN contrairement à l'ADN n'est formé que d'un seul brin. Des molécules d'ARN sont impliquées dans différents processus biologiques, et on distingue les ARN *ribosomiques* (ARNr), les ARN *de transfert* (ARNt), et les *ARN messagers* ou ARNm. Un brin d'ARN peut s'apparier avec un brin d'ADN s'ils sont complémentaires.

1.1.3 La réplication de l'ADN

L'ADN étant la molécule garante de l'information génétique, il est nécessaire de pouvoir transmettre fidèlement cette information aux descendants. Ainsi, lorsqu'une cellule "mère" se divise pour former deux cellules "filles", chacune de ces deux cellules filles doit au final comporter un génome complet. Un mécanisme, appelé la *réplication*, permet de générer à partir d'une double hélice d'ADN, deux doubles hélices, copies conformes à l'originale. Cette réplication est *semi-conservative* : chacune des deux nouvelles doubles hélices est formée de l'un des deux brins originaux. Le brin manquant est synthétisé sur le brin original, en respectant la règle d'appariement. La Figure 1.2 illustre le procédé.

1.1.4 Les protéines

Les protéines sont des macromolécules directement impliquées dans toutes les fonctions d'une cellule. L'hémoglobine par exemple, présente dans les globules rouges, permet la fixation de l'oxygène afin d'assurer son transport dans le sang. L'actine et la myosine sont deux protéines qui par leurs interactions permettent aux cellules musculaires de se contracter. D'autres protéines s'assemblent en fibres structurelles. Les enzymes sont une classe de protéines qui catalysent et contrôlent des réactions chimiques. Tout

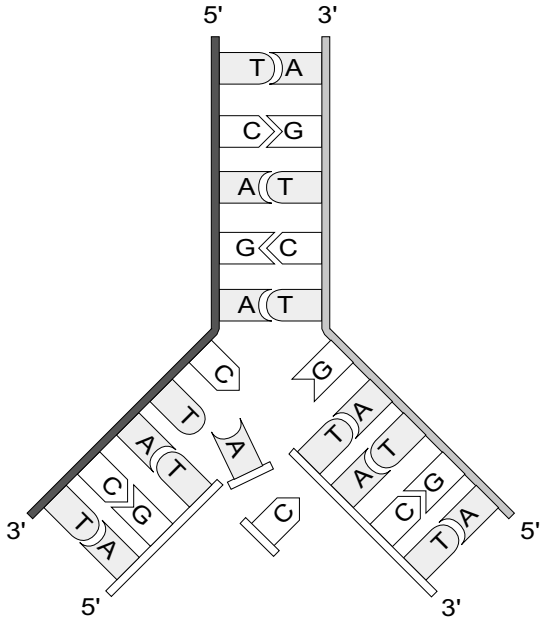


Fig. 1.2 : La réplication de l'ADN est semi-conservative. Chacune des deux copies va hériter de l'un des deux brins parents. Le nouveau brin est directement synthétisé sur le brin parent, en respectant la complémentarité des bases.

comme les acides nucléiques, les protéines sont caractérisées par un assemblage linéaire de composants de base : les *acides aminés*. Vingt acides aminés différents sont utilisés, qui peuvent présenter des caractéristiques physico-chimiques très différentes. La séquence des acides aminés constitue la structure *primaire* d'une protéine. En effet, pour être fonctionnelle, une protéine doit adopter une conformation tridimensionnelle, déterminée par un repliement donné sur elle-même. Sa fonctionnalité peut encore éventuellement dépendre d'un assemblage avec une ou plusieurs autres protéines. Toutes les protéines assurent leur fonction par des interactions spécifiques avec d'autres molécules (autres protéines, acides nucléiques, métaux, sucres, lipides...). Ces interactions spécifiques sont rendues possibles par une conformation tridimensionnelles donnée, permettant à certains acides aminés, non forcément contigus dans la séquence primaire, d'interagir avec une autre molécule.

Lorsque la séquence d'une protéine donnée est représentée sous la forme d'un texte, on utilise un code à une lettre pour chaque acide aminé (Table 1.1).

1.1.5 La synthèse des protéines

L'information de l'ordonnancement des acides aminés dans la structure primaire d'une protéine est contenue dans des régions dites *codantes* de l'ADN, régions elles-mêmes contenues dans les *gènes*. La définition du gène n'est pas aisée, mais on peut simplement dire qu'un gène est l'ensemble des nucléotides d'une molécule d'ADN impliqués dans la synthèse, et dans la régulation de cette synthèse, d'une protéine ou d'un groupe de protéines. Un gène contient donc en plus de la région codante, des régions régulatrices, qui en interaction avec des protéines spécifiques peuvent activer ou réprimer la synthèse de la protéine codée dans le gène. La séquence des ARNt et des ARNr est également codée dans des gènes spécifiques.

Code à une lettre	Code à trois lettre	Description
A	Ala	Alanine
B	Asx	Asparagine ou acide aspartique
C	Cys	Cystéine
D	Asp	Acide aspartique
E	Glu	Acide glutamique
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Méthionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Thréonine
V	Val	Valine
W	Trp	Tryptophane
Y	Tyr	Tyrosine
Z	Glx	Glutamine ou acide glutamique

Tab. 1.1 : Code à une et à trois lettres pour désigner chacun des vingt acides aminés. Les symboles 'B' et 'Z' du code à une lettre représentent chacun l'ambiguïté de deux acides aminés particuliers. L'asparagine et l'acide aspartique sont souvent confondu lors du processus de séquençage d'une protéine. Si le résultat ne peut être tranché, on représente l'acide aminé par le symbole ambigu 'B'. Il en est de même pour la glutamine et acide glutamique.

Deux étapes principales sont nécessaires pour produire une protéine à partir de l'information codante d'un gène : la *transcription* et la *traduction*.
La transcription est l'opération qui consiste à synthétiser une molécule d'ARNm complémentaire à la région codante de l'ADN. Cette opération présente des similarités avec la procédure de réplication présentée auparavant. La double hélice de l'ADN est localement ouverte afin de permettre aux nucléotides d'ARN de former des appariements complémentaires sur la région codante. L'ARNm est synthétisé par l'ajout successif de nucléotides correctement appariés. Il correspond ainsi à une copie complémentaire de la séquence de la région codante. Cet ARNm doit encore, dans le cas des eucaryotes, subir un processus de maturation, appelé *épissage*. Les régions codantes sont composée d'une alternance de régions appelées *introns* et *exons*. Seuls les exons contiennent l'information nécessaire à la synthèse de la protéine. La procédure d'épissage consiste à exciser de l'ARNm les régions correspondant aux introns de la partie codante. L'ARNm ainsi mature peut alors être traduit.
La traduction est l'étape qui consiste à assembler successivement des acides aminés dans l'ordre décrit par le "message" porté par l'ARNm. Ce message est lu par triplets conti-

Le code génétique universel

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

Tab. 1.2 : Le code génétique donne pour chacun des 64 codons possibles l'acide aminé correspondant. Notons le codon 'AUG' qui est le signal de début du message. Le premier acide aminé inséré est donc dans tous les cas une méthionine. Les codons 'UAA', 'UAG' et 'UGA' donnent le signal que le message est terminé et donc le processus de traduction doit s'arrêter.

gus de nucléotides ou *codons*. Une correspondance entre les différents codons et les acides aminés est établie par le *code génétique universel* (Figure 1.2). Son universalité, fait remarquable, est un des arguments militant en faveur d'une origine commune de la vie. Comme il y a $4^3 = 64$ codons possibles et seulement vingt acides aminés différents, plusieurs codons codent pour le même acide aminé. Cette synthèse implique une machinerie moléculaire complexe. L'ARNm est "lu" dans le *ribosome*, un complexe formé d'ARNr (ARN ribosomique) et de protéines. Des ARNt (ARN de transfert) font le lien entre les acides aminés et leur codon correspondant. La Figure 1.3 illustre le processus.

1.1.6 Les mutations et l'évolution

La phylogénie est la science des relations de filiation. Ces relations peuvent être étudiées à différents niveaux. On peut par exemple, à partir de critères morphologiques, chercher à reconstruire l'histoire évolutive d'un groupe d'espèces vivantes. On cherche alors à établir les liens de filiation qui relient ces espèces entre elles, par l'intermédiaire d'ancêtres communs disparus, mais dont certains critères morphologiques sont connus grâce à leurs fossiles.
La phylogénie moléculaire, qui nous intéresse plus particulièrement dans ce travail, étudie ces relations au niveau des molécules d'ADN, d'ARN ou de protéines, par comparaison de leurs séquences. En comparaison des études phylogéniques basées sur des critères morphologiques, on ne dispose pas, sauf dans de très rares cas, d'informations sur les séquences ancestrales.

L'évolution des séquences d'ADN se produit par une série de changements au cours

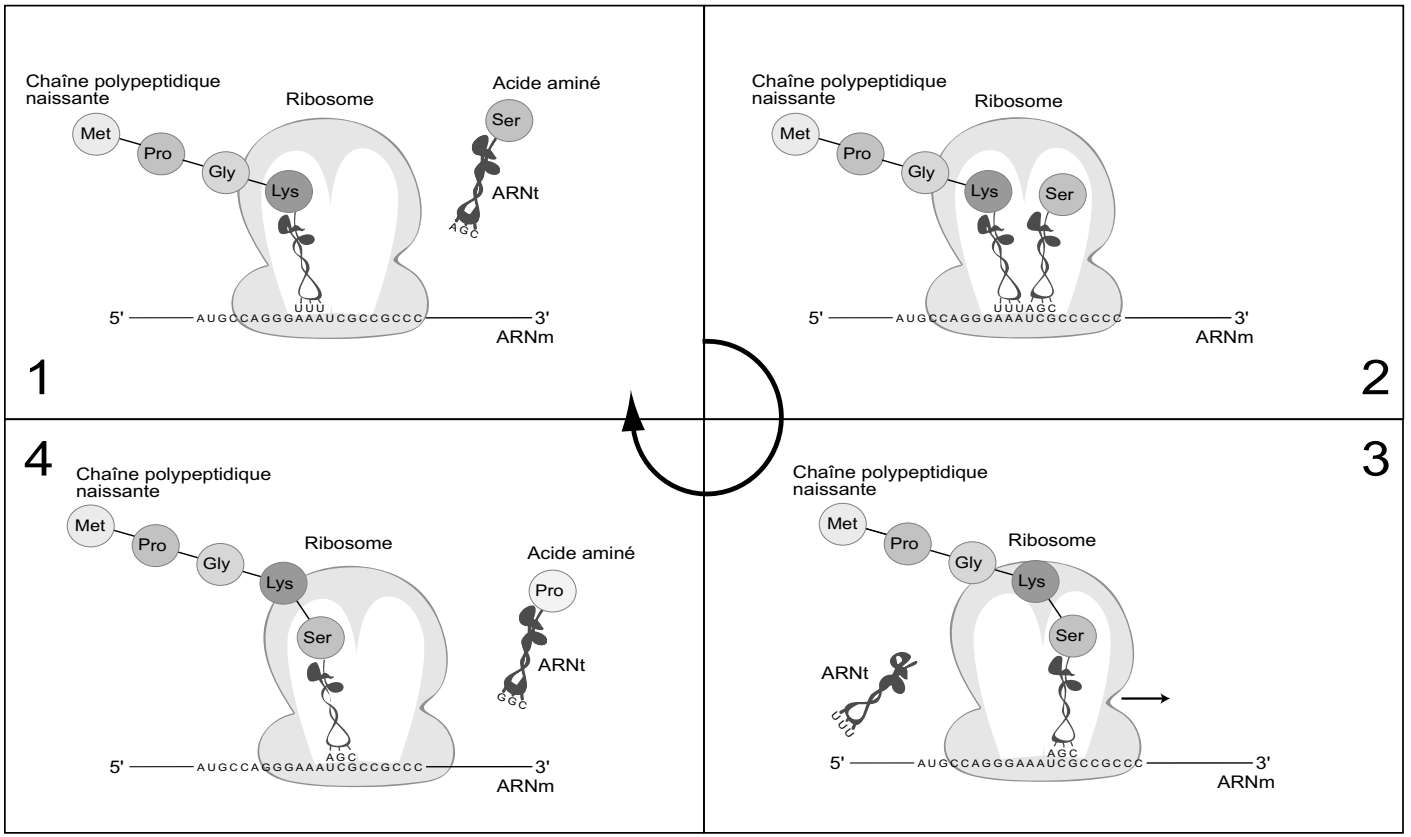


Fig. 1.3 : Traduction d'un ARNm et synthèse de la protéine correspondante. Le ribosome est un complexe composé d'ARNr et de protéines, qui permet la synthèse d'une protéine par assemblage d'acides aminés. Le décodage du message sur l'ARNm est effectué par les ARNt, qui font correspondre un codon avec un acide aminé donné. L'ARNm est lu codon par codon et les acides aminés sont successivement assemblés pour former la chaîne polypeptidique naissante. La traduction est toujours initiée par le codon 'AUG' qui code pour la méthionine. La traduction est interrompue lorsqu'elle rencontre un des trois codons 'STOP' (UAA, UAG ou UGA). (Illustration modifiée, dont l'original à été créé par Patricia Hernandez)

du temps. Les mécanismes à la base de ces changements sont les *mutations*. Une mutation est un changement, spontané ou induit, dans la séquence d'une molécule d'ADN. Dans le cas des organismes unicellulaires, comme les bactéries, les mutations sont directement transmises aux descendants. Pour les organismes multicellulaires dotés d'un mode de reproduction sexué, il est nécessaire qu'une mutation touche l'ADN contenu dans une cellule sexuelle, (ovules, spermatozoïdes), pour pouvoir être transmise à un éventuel descendant. Il existe beaucoup de types de mutations, et leur étude est une science en soi. Les causes des mutations sont diverses : les erreurs de réplication de l'ADN, les transposons (courtes séquences d'ADN qui ont la faculté de se déplacer ou se répliquer à l'intérieur d'un génome), certains virus, ou encore des agents mutagènes physiques ou chimiques. Nous décrirons les mutations principales, ou plutôt leurs effets sur la molécule d'ADN. Le lecteur intéressé par les mécanismes moléculaires qui les produisent trouvera des descriptions détaillées dans (Griffiths et al., 2002).
On peut distinguer les mutations *géniques* et les mutations *chromosomiques*. Les premières n'impliquent qu'un petit nombre de paires de bases, alors que les deuxièmes provoquent des modifications plus importantes d'un ordre allant de la centaine à plusieurs millions de paires de bases.
Les mutations géniques sont les *substitutions*, les *additions* (également appelée *insertions*), et les *délétions*. Une substitution est le remplacement d'une paire de bases par une autre. Sans détailler les différents types de substitutions, leurs effets provoquent un changement ponctuel dans la séquence de l'ADN. Ce changement est souvent sans conséquences, sauf s'il intervient dans une région codante. Il peut alors avoir un effet direct sur la séquence de la protéine codée. Un codon codant pour un acide aminé peut être modifié en un autre codon représentant un acide aminé différent. Lorsque le nouveau codon code pour le même acide aminé que l'ancien, on parle de mutation *silencieuse*. Une conséquence plus importante peut être provoquée par la création d'un codon 'STOP' à l'intérieur de la région codante ce qui provoquera la fin prématurée de la synthèse de la protéine.
Les additions et délétions suppriment ou ajoutent une paire ou plusieurs paires contiguës de nucléotides dans la séquence. Lorsqu'elles se produisent dans une région codante, elles peuvent avoir un effet désastreux. Si le nombre de paires de nucléotides insérés ou supprimés n'est pas un multiple de trois, il en résultera un décalage du cadre de lecture, provoquant la modification de tous les acides aminés dont le codage se trouve en aval de la mutation.
Il existe également plusieurs types de mutations chromosomiques : les *translocations* consistent en l'échange de grandes régions entre deux chromosomes, les *délétions* provoquent la perte totale d'une région chromosomique, les *inversions* consistent en un retournement complet d'une région, et les *duplications* copient une région à un autre endroit du génome. Ces dernières fournissent un excédent de matériel génétique qui peut évoluer vers de nouvelles fonctions. Ainsi, un gène complet peut être copié, puis par modifications au cours de l'évolution, diverger et finalement coder pour deux protéines différentes *homologues* car elles sont issues de la même séquence ancestrale. On retrouve ainsi, au sein d'une même espèce et entre des espèces différentes des familles de protéines homologues.

Les petites mutations (mutations ponctuelles) sont dans la majorité des cas sans conséquences, provoquant de petites variations dans l'information génétique des individus d'une espèce. En revanche, si ces mutations touchent un gène, elles peuvent avoir des conséquences importantes sur la protéine concernée ou sur sa régulation, se répercutant ainsi directement sur la cellule ou l'organisme. S'il en résulte un dysfonctionnement total de la protéine, il est fort probable que la cellule ou l'organisme ne puisse pas se

développer et assurer sa survie. On parle dans ce cas de mutation létale, et elles ne sont donc par conservés dans l'évolution. Si au contraire, le changement induit n'a pas de conséquences pénalisantes, la mutation a une chance d'être transmise aux descendants, pouvant éventuellement induire des variations fonctionnelles, morphologiques ou comportementales, lesquelles sont à la base de l'évolution des espèces. Un exemple classique est illustré par une classe de gènes communs au monde animal : les gènes *homéobox* impliqués dans le développement des embryons. Des expériences menées sur les mouches drosophiles ont montré que des mutations affectant ces gènes peuvent être la cause de modifications morphologiques importantes (Carroll, 1995).

Ainsi, la comparaison de séquences d'ADN ou de protéines, au sein d'une même espèce ou de différentes espèces permet de spéculer sur les mutations qui se sont produites au cours de l'évolution, ainsi que sur la nature des séquences ancestrales probables. La procédure en bioinformatique qui cherche à expliciter ces mécanismes à partir de séquences supposées homologues est l'alignement de séquences.

1.2 Motivations

L'alignement multiple de séquences est une problématique très étudiée en bioinformatique. Il existe d'une manière générale deux approches pour ce problème, chacune étant dédiée à un type d'alignement multiple particulier. L'approche la plus connue des biologistes, et la plus ancienne est basée sur des algorithmes gloutons, qui dans la grande majorité des cas construisent l'alignement sur la base d'informations obtenues par une comparaison deux à deux des séquences. L'autre approche, un peu plus récente, utilise généralement des algorithmes statistiques d'optimisation, qui considèrent toutes les séquences simultanément.
Les approches gloutonnes permettent d'aligner les séquences sur toute leur longueur, et peuvent prendre en considération les événements mutationnels d'insertion et de délétion. Elles sont de plus bien adaptées à l'alignement de protéines, car le système de score qu'elles utilisent permet de prendre en considération la nature des acides aminés, et peuvent ainsi évaluer la pertinence biologique d'un appariement d'acides aminés différents.
Réciproquement, les approches d'optimisation statistiques sont en règle générale plus spécifiques, par le fait qu'elle sont dédiées à l'alignement local, c'est à dire à de courtes régions contiguës, et permettent de prendre en considération des événements mutationnels comme les duplications. Elle peuvent aussi généralement ignorer certaines séquences de la procédure si aucune de leurs régions ne sont jugées significatives dans l'alignement. Elles ne prennent par contre pas en considération les événements mutationnels d'insertion et de délétion.

Deux différences majeurs entre les deux approches peuvent être relevées. Premièrement, le fait que les approches statistiques considèrent toutes les séquences simultanément au cours de l'optimisation les rend capables de détecter des similarités beaucoup plus faibles, qui ne seraient pas significatives par une comparaison deux à deux des séquences. L'autre différence concerne le système de score utilisé. Les méthodes statistiques utilisent une mesure basée sur la distribution des symboles dans les colonnes de l'alignement. Cette mesure peut s'interpréter comme un degré de conservation dans la colonne. Plus le nombre de symboles identiques est élevé et plus le score est important. Cependant, un point critique de cette approche est que les symboles sont considérés totalement indépendamment

les uns des autres. Si les conséquences de cette observation ne sont pas très importantes pour les séquences nucléiques, elles signifient dans le cas des protéines, que la propriété physico-chimique de leurs acides aminés n'est pas prise en compte. Or certains acides aminés partagent des propriétés très similaires, rendant la substitution de l'un par l'autre dans une protéine sans grande conséquence. Ce système de score n'est donc pas capable de considérer les appariements d'acides aminés différents qui sont biologiquement pertinents. Ce problème à été relevé à plusieurs reprises, mais aucune solution n'a à notre connaissance été apportée. En conséquent, la grande majorité des programmes développés sous ces approches sont dédiés à l'alignement de séquences nucléiques et en particulier de sites de fixation de facteurs de transcription.

Sur la base de ces observations, il nous a paru important de proposer aux biologistes, une méthode permettant de produire des alignements multiples fiables de séquences protéiques distantes. Il était alors évident que ce problème devait être approché par une méthode qui considère toutes les séquences simultanément. Étant donné que la nature du problème est fondamentalement combinatoire, nous avons décidé de le définir sous la forme d'un problème d'optimisation par voisinage, car ce point de vue nous a paru le plus adéquat pour pouvoir étudier et comprendre la structure du problème, de façon à pouvoir lui apporter des solutions d'optimisation pertinentes. La fonction objectif classiquement utilisée sur ce problème ne prenant pas en compte la nature des acides aminés, il était alors essentiel de développer une solution à ce problème, en définissant une nouvelle fonction mesurant le degré de conservation d'une colonne, mais avec la possibilité de prendre en compte une mesure d'interchangeabilité des acides aminés. Une telle fonction objectif, couplée à une stratégie d'optimisation tirant parti du maximum d'informations préalables sur le problème nous à paru être la meilleur approche pour le but que nous nous étions fixé.

1.3 Organisation du document

Cette thèse est organisée de la façon suivante. Le Chapitre 2 présente un panorama des méthodes génériques d'optimisation combinatoire par voisinage. Nous commençons par définir les notions de base permettant de caractériser un problème d'optimisation, à savoir l'espace de recherche, le paysage d'exploration, le voisinage, les maximums locaux et globaux et les bassins d'attraction. Nous décrivons également les principales approches d'optimisation génériques, à savoir les grimpeurs et les approches évolutionnistes.
Le Chapitre 3 donne l'état de l'art des programmes et méthodes couramment utilisés pour l'alignement de séquences. Nous discutons également des fonctions de scores, ou fonctions objectif, qui sont utilisées pour mesurer la qualité biologique des alignements produits.
Le Chapitre 4 présente notre approche originale pour le problème de l'alignement local multiple et sans indels. Nous approchons ce problème sous la forme d'une optimisation combinatoire par voisinage, en définissant les espaces de recherche, les fonctions de voisinages, et différentes stratégies d'optimisation. Nous décrivons également dans ce chapitre une nouvelle fonction objectif dédiée à l'optimisation d'alignements multiples de séquences de protéines. Le Chapitre 5 présente une série de résultats, impliquant des données réelles et artificielles. Nous y évaluons l'efficacité de nos stratégies, en comparaison avec différents programmes de référence. Nous y comparons également notre nouvelle fonction objectif avec celle classiquement utilisée, sur sa capacité à identifier des alignements de séquences protéiques faiblement conservées. Nous évaluons également ces fonctions par rapport aux

effets qu'elles peuvent apporter sur le paysage d'exploration. Finalement, le Chapitre 6 présente nos conclusions et les principales contributions scientifiques de notre travail, ainsi que les perspectives dégagées.

Chapitre 2

L'optimisation combinatoire par voisinage

2.1 Introduction

L'optimisation combinatoire est une problématique consistant, pour un ensemble de variables prenant des valeurs entières, à déterminer leurs instanciations, éventuellement sous contraintes, correspondant au maximum (ou minimum) d'une fonction de ces variables, OF, généralement appelée *fonction objectif*. L'ensemble des combinaisons pouvant être prises par ces variables constitue *l'espace de recherche* du problème. Une combinaison donnée est appelée une *solution potentielle* ou plus simplement une *solution*, alors que les instanciations des variables qui maximisent OF sont appelées les *solutions optimales* du problème. Dans le cas d'une optimisation combinatoire *sous contrainte*, on définit un ensemble de contraintes sur les variables rendant non valide un sous-ensemble de l'espace de recherche.
Lorsque la fonction objectif OF n'est pas connue analytiquement, c'est à dire que l'on ne peut pas l'exprimer directement à partir des variables, on dit que le problème est de type "boîte noire". Dans ce cas, on ne peut connaître la valeur de la fonction que de façon indirecte pour chacun des points de l'espace de recherche. On ne dispose donc pas de propriétés sur OF et les méthodes d'optimisation par gradient ou par "fitting" sont inutilisables. Ce chapitre est essentiellement dédié au problèmes de type "boîte noire".

Les méthodes d'optimisations sont très nombreuses. Il existe plusieurs classifications possibles, l'une d'elle consiste en la séparation en deux groupes : les méthodes par voisinage et les méthodes par construction.
Les méthodes par voisinage structurent l'espace de recherche en un ou plusieurs *paysages* par l'intermédiaire de fonctions de *voisinage*. Ces méthodes disposent en permanence d'une ou plusieurs solutions complètes (pour lesquelles on a calculé la valeur de OF), qu'elles modifient de façon plus ou moins importante pour obtenir de nouvelles solutions. Nous décrirons ce type de méthodes plus en détail dans le reste de ce chapitre.
L'autre groupe de méthodes, dites par construction, élabore une solution en instanciant successivement les variables. Certaines de ces méthodes sont dites exactes dans le sens ou elles garantissent de trouver la solution optimale. La plus simple d'entre elles est l'exploration exhaustive de l'espace de recherche par arborescence comme le fait l'algorithme de "retour en arrière" par exemple (Russel and Norvig, 2003). La complexité moyenne de

ces approches est exponentielle avec le nombre de variables. Des variantes plus ou moins complexes comme les algorithmes A^* ou de *branch and bound* utilisent des heuristiques exactes pour élaguer l'arbre de recherche. La complexité de ces méthodes est directement dépendante de l'heuristique utilisée et de son adéquation avec la fonction objectif. Un exemple d'algorithme de branch and bound pour le problème de l'alignement local multiple sans indels est décrit dans (Horton, 2001).
La plupart des autres méthodes par construction ne garantissent de trouver la ou les solutions optimales que dans certaines conditions liées aux propriétés du problème considéré. La stratégie non-exacte la plus simple est la construction gloutonne dans laquelle on instancie les variables les unes après les autres, selon la valeur potentielle de la solution non complètement instanciée. Une instanciation effectuée ne peut pas être remise en cause par la suite. Une méthode gloutonne plus élaborée est la *recherche par faisceau* (beam search). Cette méthode parcourt et élague un arbre de recherche. Elle doit disposer d'un score pour les noeuds internes correspondant à des solutions non complétement instanciées. L'arbre est parcouru par niveaux. Tous les noeuds d'un niveau sont considérés, puis on ne conserve que les k meilleurs, les autres étant élagués avec leurs sous-arbres correspondants, en prenant bien sûr le risque d'élaguer un sous-arbre menant à la solution optimale. On progresse ainsi niveau par niveau jusqu'aux feuilles, correspondant à des solutions complètes. A chaque niveau, on évalue donc $k * n$ solutions incomplètes, où n est le nombre moyen de fils par noeuds. Le paramètre k est appellé la *largeur du faisceau*. Sa valeur influe directement sur les complexités spatiales et temporelles. Cette technique est utilisée par la méthode d'alignement multiple CONSENSUS (Hertz et al., 1990), que nous décrivons dans la Section 3.5.4.

La majeure partie des problèmes d'optimisation combinatoire ont été démontrés NP-complets et donc particulièrement difficiles à résoudre. Des méthodes exactes (programmation dynamique, branch and bound...) peuvent s'avérer efficaces dans certaines instances de ces problèmes, mais le plus souvent aucune solution exacte ne peut être découverte en un temps raisonnable et des approches heuristiques non-exactes doivent être utilisées pour proposer des solutions approchées de bonne qualité.

2.2 Les métaheuristiques

Les métaheuristiques sont une classe des méthodes d'optimisation combinatoire. Ce terme est apparu dans les années quatre-vingt. Son origine est probablement issue d'une publication sur la recherche taboue (Glover, 1986). Dans sa signification première, une métaheuristique désignait une méthode d'exploration qui regroupait plusieurs heuristiques. Ce terme à été par la suite largement repris au delà de sa définition première sans pour autant être clairement redéfini. Les métaheuristiques regroupent beaucoup de méthodes par voisinage, telle que les *grimpeurs*, le *recuit simulé*, les *algorithmes évolutionnistes*, mais par contre, le *simplex* (méthode par voisinage) en est exclu. Les méthodes d'*intelligence en essaim*, approche par construction, font également partie des métaheuristiques. Par contre, il n'y a pas de consensus pour le *branch and bound*. De manière générale, les techniques de recherche opérationnelle en sont clairement exclues. Il parait alors difficile de définir les métaheuristiques sans énumérer les méthodes qui en font partie. Une solution serait de dire que les métaheuristiques regroupent les méthodes d'optimisation combinatoire dédiées aux problèmes de type "boîte noire". Le lecteur intéressé pourra se référer à (Hao et al., 1999) qui présente une description complète sur les métaheuristiques.

2.3 L'espace de recherche

Soit $V = [v_1, v_2, \ldots, v_n]$ un vecteur de n variables binaires[1]. Une instanciation de V définit une coordonnée dans un espace discret à n dimensions. Soit X l'ensemble des 2^n instanciations possibles de V. Chaque élément x correspond donc à un point de l'espace précité, et pour cette raison, l'ensemble X est appelé l'*espace de recherche* du problème. Nous pourrons par la suite nous référer à la i^{eme} dimension d'un élément x pour parler de v_i, sa i^{eme} coordonnée. Un élément x est également appelé une *solution potentielle*.
Soit une fonction $OF : X \longrightarrow \mathbb{R}$ dénommée *fonction objectif* ou *fonction de fitness*. Un problème d'optimisation combinatoire est un problème pour lequel on cherche le ou les maximums de OF pour tous les points de l'espace de recherche X. On dit que ce problème est de taille n, soit le nombre de variables qui le compose. Sa complexité est le nombre minimum de points de X pour lesquels OF doit être calculée afin de garantir la découverte du ou des maximums.

Soit M l'ensemble des solutions optimales ou *maximums globaux* définit par :

$$M = \{a_m \in X \mid \forall a_i \in X, OF(a_m) \geq OF(a_i)\}$$

c'est à dire l'ensemble des éléments de X pour lesquels OF est maximale. Une *métaheuristique* est un algorithme générique qui explore un sous-ensemble Ex de X et produit en résultat S, l'ensemble des éléments définis par :

$$S = \{a_s \in Ex \mid \forall a_i \in Ex, OF(a_s) \geq OF(a_i)\}$$

Les problèmes considérés étant le plus souvent NP-complets, on ne peut garantir de trouver les solutions M en un temps polynomial sur n. Les métaheuristiques ne sont en général pas des algorithmes exacts et n'offrent donc pas la garantie de trouver la solution optimale. Elles cherchent plutôt à trouver une solution de "bonne qualité" en un temps raisonnable.

2.4 Un modèle de paysage

Les méthodes d'optimisation par voisinage échantillonnent l'espace de recherche en utilisant une ou des représentations structurées appelées *paysages*. Dans le cas le plus simple (et le plus fréquent), les "déplacements" sont effectués par des opérateurs qui transforment directement un élément de X en un autre. Mais des transformations dans des structures plus complexes sont également utilisées par des méthodes classiques telles que les algorithmes génétiques (décrit dans la Section 2.6.2). Nous définirons également les notions de *maximum local* et de *bassin d'attraction*, lesquelles sont directement liées à un paysage.
Nous verrons que ces notions ne sont pas liées au problème, mais à une manière de conduire son optimisation et donc à une fonction de voisinage ainsi qu'à la fonction objectif. Ainsi, une stratégie d'optimisation qui utilise plusieurs fonctions de voisinage définit autant de paysages et parler de maximums locaux ne peut se faire qu'en fonction d'un paysage donné.

Nous conseillons la lecture de la thèse de Terry Jones (Jones, 1995) qui définit ces notions sous un point de vue probabiliste.

[1] Les problèmes classiques d'optimisation combinatoire ainsi que la majorité des études qui sont faites se rapportent à des ensembles de variables binaires. Il est cependant courant d'utiliser des variables pouvant prendre plus de deux valeurs.

2.4.1 Un modèle de paysage

Nous allons définir le prototype d'un paysage simple, c'est à dire un paysage résultant de l'exploration d'un seul point de X à la fois. Un tel paysage est associé à une fonction de voisinage de type $\phi : X \longrightarrow P(X)$, ou $P(X)$ est l'ensemble des parties de X. La notation $\phi(a) = B$ avec $a \in X$ et $B \subseteq X$, indique que B est l'ensemble des points de X voisins du point a et $\phi(a)$ est appelé le *voisinage* de a. Une fonction de déplacement ou *opérateur* $D_\phi : X \longrightarrow X$, est une fonction qui choisit (stochastiquement ou non) un élément $b \in \phi(a)$. On parle dans ce cas d'un déplacement de a vers b. Un *paysage* $G(X, \phi, OF)$ est une organisation topologique des éléments de X implicitement définie par ϕ et OF. Cette organisation correspond à un graphe dirigé où chaque élément de X est représenté par un noeud g_i et chaque noeud est relié par un arc directionnel à chacun des noeuds faisant partie de son voisinage. Nous utiliserons la notation $\phi(g_i)$ pour parler de l'ensemble des noeuds sur lesquels arrive un arc issu de g_i. A chaque noeud g_i est en plus associée une valeur, appelée *hauteur*, laquelle correspond dans le cas de ce modèle simple à $OF(x)$ la valeur de fitness du point x associé au noeud g_i. Du point de vue d'un processus d'exploration, $\phi(g_i)$ correspond à l'ensemble des noeuds "visibles" ou accessibles depuis g_i. Un exemple extrêmement courant est le voisinage de Hamming. La distance de Hamming entre deux points a et b correspond au nombre de variables instanciées différemment entre ces deux points, par exemple si $a = [1111]$ et $b = [1010]$, la distance de Hamming entre a et b vaut 2. Le voisinage de Hamming d'un point a correspond à l'ensemble des points de X à une distance de Hamming de 1. La figure 2.1 représente la topologie complète d'un paysage de Hamming pour des vecteurs binaires de taille quatre.

Ce modèle simple suffit à décrire la majorité des paysages découlant d'une optimisation de type grimpeur (que nous abordons plus loin), il faut cependant garder à l'esprit que des paysages plus complexes peuvent également être utilisés. Dans le cas des algorithmes génétiques (Section 2.6.2) par exemple, il existe des opérateurs de déplacement qui produisent deux points à partir de deux points. Dans ce cas, chaque noeud du graphe représente un couple de points et la valeur de hauteur associée correspond à une fonction des deux valeurs objectif (typiquement le maximum ou la moyenne). La Figure 2.2 présente une partie de la topologie du paysage résultant de l'opérateur de recombinaison à un point (opérateur classique des algorithmes génétiques) pour des vecteurs binaires de taille quatre.

2.4.2 Les maximums locaux

La notion de *maximum local* est dépendante d'un paysage. Un maximum local m est un noeud de G tel que :

$$\forall g_i \in \phi(m), OF(m) \geq OF(g_i)$$

Un maximum local est donc un noeud, pour lequel l'ensemble de ses voisins p_i ont une hauteur $OF(p_i)$ plus petite ou égale à sa hauteur. Il faut noter que les maximums globaux M sont des éléments de X, alors qu'un maximum local est un noeud de G. Dans tous les cas, les points de M sont présent dans un des maximums locaux, quelle que soit ϕ, M ne dépendant pas d'une fonction de voisinage.

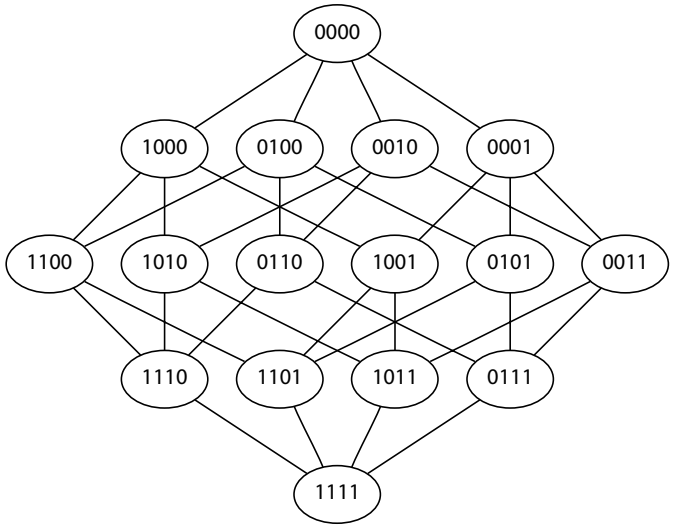


Fig. 2.1 : Topologie du paysage de Hamming pour des vecteurs binaires de taille quatre. Ce paysage très courant ne contient qu'un seul point de l'espace de recherche par noeud. Chaque vecteur v_i a exactement quatre voisins, correspondant à toutes les façons de produire un vecteur à une distance de Hamming de 1 de v_i. Le graphe n'est pas dirigé car une opération de Hamming est symétrique.

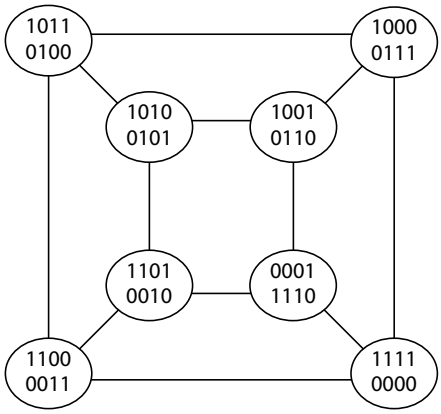


Fig. 2.2 : Une partie de la topologie du paysage de recombinaison 1-point pour des vecteurs binaires de taille quatre. Le voisinage de recombinaison produit un ensemble de couples de points à partir d'un couple de points. Ce type d'opérateur est utilisé par les algorithmes génétiques, lesquels sont décrits dans la Section 2.6.2. Le graphe de paysage complet aurait $2^4 \cdot 2^4 = 256$ noeuds et se présenterait sous forme d'îlots disjoints. L'ensemble de tous les points possibles de l'espace de recherche sont néanmoins représentés dans cet îlot, soit 2^4 vecteurs. Le graphe n'est pas dirigé car l'opérateur de recombinaison est symétrique.

2.4.3 Les bassins d'attraction

Une liste de noeuds parcourus successivement par un opérateur D_ϕ constitue un chemin $\pi = (g_1, g_2, \ldots, g_n)$ tel que :

$$g_i = D_\phi(g_{i-1}), \forall i \in \{2, 3, \ldots, n\}$$

En d'autres termes, un opérateur de déplacement qui utilise son résultat précédent pour une nouvelle opération "chemine" dans le paysage. Soit la fonction booléenne $\pi D_\phi(g_i, g_j)$ telle que :

$$\pi D_\phi(g_i, g_j) = \begin{cases} 1 & \text{si un chemin de } g_i \text{ vers } g_j \text{ existe} \\ 0 & \text{sinon} \end{cases}$$

Un *bassin d'attraction* $B(m, D_\phi)$ d'un maximum local m est l'ensemble des noeuds de G, à partir desquels il existe un chemin vers m.

$$B(m, D_\phi) = \{g_i \mid \pi D_\phi(g_i, m) = 1\}$$

2.4.4 La complexité des problèmes

De manière générale, une métaheuristique générique sera peu performante sur un problème donné. Le théorème "no free lunch" (Wolpert and Macready, 1997) a montré que la performance de n'importe quel algorithme par voisinage moyennée sur tous les paysages possibles est la même que celle d'un algorithme de recherche aléatoire. Même si l'on peut contester la validité pratique de ce théorème, il en découle toutefois qu'il est illusoire d'espérer trouver un algorithme générique résolvant efficacement tous les problèmes d'optimisation combinatoire. Chaque problème, voire chaque instance d'un problème a sa propre structure et sa propre complexité, pour lesquelles des fonctions de voisinage génériques ont de fortes chances d'être inadaptées.
La complexité d'un problème d'optimisation combinatoire peux s'exprimer sous plusieurs points de vue. Dans le cas de grimpeurs par exemple, on peut l'exprimer en terme de nombre de maximum locaux, de taille des bassins d'attraction, et de nombre d'opérations moyen requis pour atteindre la solution optimale en supposant que l'on parte d'un point faisant partie du bassin d'attraction de celle-ci. Une autre façon plus élaborée d'aborder la complexité d'un problème est de considérer les dépendances entre les variables qui le composent ou son degré d'*épistasie*. Cette problématique est abordée en détail dans le mémoire d'habilitation de Robin Gras (Gras, 2004). Afin de décrire la notion de dépendance entre variables, nous nous intéressons au cas où la fonction objectif OF peut être décomposée en plusieurs sous-fonctions OF_i. Si la fonction OF de taille n peut être décomposée en n/k sous-fonctions indépendantes de taille k, la complexité n'est plus de 2^n mais de $2^k n/k$ car le problème peut être résolu en optimisant indépendamment les n/k sous-problèmes qui le composent (si aucune autre propriété du problème ne peut être exploitée réduisant encore sa complexité). Dans la situation où $k = 1$, c'est à dire que les n variables du problème sont indépendantes les unes des autres, la complexité est linéaire en n. La fonction OneMax, qui renvoie la somme des variables, illustre ce cas particulier, chaque variable étant totalement indépendante des autres. Le degré d'épistasie d'un problème correspond au nombre maximum de variables dont dépend, dans le calcul de OF, chacune des n variables. Par exemple, un problème de degré d'épistasie 0 est un problème où toutes les variables sont indépendantes. L'épistasie est un concept plus général que la décomposabilité d'une fonction puisque un degré d'épistasie k n'implique pas forcément que le problème

soit décomposable en n/k sous-problèmes complètement indépendants. En effet, les sous-problèmes peuvent contenir des variables communes et on peut, par exemple, tout à fait avoir n sous-problèmes de taille k.

2.5 Les grimpeurs

On appelle *grimpeur* un type de métaheuristique qui explorent un chemin π unique par une succession d'opérations D_ϕ ascendantes (ou à tendance ascendante) dans le paysage $G(X, \phi, OF)$. En l'absence de connaissances préalables sur un problème, les grimpeurs sont souvent utilisés par défaut avec le voisinage de Hamming. Cependant, le principe du grimpeur peut être appliqué à d'autres paysages. Nous définirons dans les prochaines sections les types courants de grimpeurs, à savoir les grimpeurs stricts, la recherche taboue, les grimpeurs probabilistes et le recuit simulé.

2.5.1 Les grimpeurs stricts

Le terme d'algorithme *grimpeur strict* provient du fait que ce type d'optimisation utilise des opérations strictement ascendantes dans le paysage. Les grimpeurs stricts sont en général de nature déterministe, sauf cas spéciaux, lorsqu'il s'agit de lever une ambiguïté par exemple. Citons notamment *le plus grand ascendant*, dont l'opérateur D_ϕ se définit par :

$$D_\phi(a) = b_i \mid b_i \in \phi(a), OF(b_i) > OF(a), \forall b_j \in \phi(a) OF(b_i) \geq OF(b_j)$$

C'est à dire qu'à chaque déplacement, on choisit toujours le noeud $b \in \phi(a)$ qui est le plus haut. L'*ascendant quelconque* est un type de grimpeur pour lequel l'opérateur D_ϕ choisit le premier élément rencontré $b \in \phi(a)$ tel que $OF(b) > OF(a)$. Les éléments de $\phi(a)$ étant considérés dans un ordre prédéfini ou au hasard.
Pour conclure, *le plus petit ascendant* dont l'opérateur de déplacement est définit par :

$$\begin{aligned} D_\phi(a) = \quad & b_i \mid b_i \in \phi(a), OF(b_i) > OF(a), \\ & \forall b_j \in \{b_k \in \phi(a) \mid OF(b_k) > OF(a)\}\ OF(b_i) \leq OF(b_j) \end{aligned}$$

b_i correspond donc au noeud de $\phi(a)$ qui a une hauteur strictement supérieure à $OF(a)$, mais qui est la plus petite possible.
Si pour ces trois grimpeurs, la fonction D_ϕ ne peut plus produire de solution (si aucun élément de $\phi(a)$ n'a de hauteur strictement supérieur à celle de a), le grimpeur se trouve alors sur un maximum local. On dit alors que le grimpeur a convergé. Beaucoup d'extensions ou de variantes existent, comme par exemple la possibilité de permettre à un grimpeur de se déplacer sur des plateaux pendant un nombre maximum d'itérations donné.
En pratique, un grimpeur strict est lancé sur plusieurs noeuds de départ appelés *graines*. On parle dans ce cas de grimpeur à départs multiples. En effet, à moins de disposer d'une fonction de voisinage parfaite, où tous les noeuds du paysage font partie du bassin d'attraction du noeud contenant le maximum global, un seul processus de grimpeur sera généralement insuffisant pour atteindre une solution satisfaisante.

2.5.2 La recherche taboue

Une variante plus sophistiquée de grimpeur strict qui a suscité une importante littérature est la *recherche taboue* (tabu search) (Barnes et al., 1995; Battiti and Tecchiolli, 1994; Glo-

ver, 1986). La base de cette approche est la même que celle des grimpeurs stricts, à la différence que le processus ne s'arrête pas forcément sur un maximum local. Le meilleur voisin peut tout de même être accepté même si sa valeur est plus petite que le point de provenance. Cependant, cette stratégie peut entraîner des cycles. Afin de les éviter, on mémorise une liste, dite taboue, des dernières variables ayant été modifiées par les étapes précédentes. Cette liste est ensuite utilisée pour interdire de modifier les variables correspondantes pour le prochain mouvement à effectuer. Elle restreint donc le voisinage du point courant à tous ceux qui ne diffèrent du point courant que pour les variables non-membres de la liste taboue. Cette liste est de taille limitée et les événements situés plus loin dans le passé que la taille de la liste sont oubliés. Cette taille est souvent qualifiée d'*horizon*. Il existe de nombreuses extensions à cette méthode comme la possibilité de faire varier la taille de la liste au cours de la recherche, ou comme l'aspiration, un procédé qui permet de lever le caractère tabou d'un point du voisinage si celui-ci possède la plus haute valeur découverte jusqu'à présent.

2.5.3 Les grimpeurs probabilistes

Les grimpeurs probabilistes fonctionnent en associant à chaque voisin une probabilité d'être choisi, cette probabilité étant évidement plus forte pour les voisins dont la valeur de la fonction objectif est meilleure. Ce type de grimpeur n'est pas strictement ascendant et garde donc des chances, lors d'une seule optimisation, de s'échapper d'un éventuel maximum local. L'opérateur D_ϕ choisit un point dans le voisinage, en effectuant un tirage aléatoire biaisé en faveur des meilleurs. Chaque voisin à une probabilité propre d'être choisi. Le calcul de cette probabilité peut se faire de différentes manières mais doit en règle générale être fortement biaisé en faveur des déplacement ascendants pour pouvoir produire un résultat satisfaisant. Ce type de grimpeur doit disposer d'une condition d'arrêt explicite, qui peut être par exemple un nombre minimum d'itérations associé avec un nombre maximum d'itérations sans amélioration.
Le programme d'alignement local multiple Gibbs Site Sampler (méthode statistique) correspond à un grimpeur probabiliste. Nous le décrivons dans la Section 3.5.1.

2.5.4 Le recuit simulé

Le *recuit simulé* (simulated annealing) (Kirkpatrick et al., 1983) est certainement la forme de grimpeur probabiliste la plus connue. Son fonctionnement est inspiré du phénomène physique de cristallisation. Lorsqu'un métal en fusion se refroidi, les molécules qui le composent vont petit à petit se structurer selon une conformation de basse énergie. Le recuit simulé commence sont exploration par un cheminement proche de l'aléatoire (température haute), puis au cours des itérations, sa fonction de déplacement est graduellement modifiée pour tendre de plus en plus vers des déplacements ascendants, en analogie avec le métal qui se refroidi petit à petit. Ce principe est généralement présenté du point de vue de la minimisation d'une fonction (énergie basse), mais dans la continuité de notre présentation, nous le décrivons comme un algorithme de maximisation.
Le principe du recuit simulé est le suivant : l'opérateur de déplacement $D_\phi(a) = b$ choisit b aléatoirement dans le voisinage de a (chaque voisin ayant la même probabilité d'être choisi). Ensuite on calcule $\delta = f(b) - f(a)$. Si la valeur δ est positive (déplacement ascendant), on accepte b, sinon on l'accepte avec une probabilité $P(t)$, dépendante du nombre

d'itération déjà écoulé, selon :

$$P(t) = e^{\frac{\delta}{g(t)}}$$

avec $g(t)$ une fonction décroissante sur t.
Le comportement global du recuit simulé varie d'une stratégie proche d'un cheminement aléatoire en début d'exploration, pour se transformer progressivement en une stratégie d'ascendant quelconque.

2.6 Les algorithmes évolutionnistes

2.6.1 Introduction

Le concept d'algorithme évolutionniste a près de quarante ans d'existence et comporte de nombreuses formes dont les plus connues sont : la programmation évolutionniste (Fogel et al., 1966), les algorithmes génétiques (Holland, 1975), les stratégies évolutives (Rechenberg, 1973) et la programmation génétique (Koza, 1992). Parallèlement aux développements continus sur ces approches classiques, de nouvelles approches basées sur une modélisation probabiliste de la population sont apparue. Citons notamment les algorithmes à distribution marginale univariées (UMDA : "univariate marginal distribution algorithm") (Mühlenbein and Voigt, 1995), et les algorithmes génétiques par construction de modèles probabilistes (PMBGA : "probabilistic model-building genetic algorithm") (Pelikan et al., 1999; Muhlenbein and Mahnig, 2001).

D'une façon générale, les algorithmes évolutionnistes suivent tous la même procédure. On crée au départ un échantillon de taille fixée de points de l'espace de recherche. Ensuite deux étapes se succèdent : La *sélection* et la *reproduction*. La sélection consiste essentiellement à supprimer certains éléments et à en dupliquer d'autres, la taille de l'échantillon étant généralement conservée. Cette opération est stochastique et favorise la duplication d'éléments de valeur de fitness haute en regard du reste de l'échantillon. La reproduction consiste ensuite à générer un nouvel échantillon à partir des éléments de l'échantillon courant. Les nouveaux éléments sont créés à partir de solutions partielles ou de particularités présentes dans la population courante. Comparé aux grimpeurs (grimpeurs simples, recuit simulé, recherche taboue) qui explorent l'espace de recherche "en solitaire", ne considérant qu'une seule solution à la fois, les algorithmes évolutionnistes cherchent à tirer parti d'une statistique sur l'échantillon des solutions considérées à un moment donné. On parle alors de *coopération* car les solutions courantes interagissent pour générer les nouvelles solutions. Ce principe repose sur le fait que cette interaction (coopération) doit être constructive et qu'une partie des solutions nouvellement créées auront une valeur de fitness plus élevée que les solutions précédentes. Ces deux étapes de sélection et de transformation sont itérées jusqu'à une condition d'arrêt fixée.

Nous décrirons le principe général des algorithmes évolutionnistes classiques à partir de l'algorithme génétique simple (AGS) (Goldberg, 1989). Nous parlerons ensuite du concept d'épistasie, ou de dépendance entre les variables du problème, lequel est un des aspects de la complexité d'un problème d'optimisation. Pour finir, nous parlerons succinctement de BOA (Pelikan et al., 1999; Pelikan, 2002), un algorithme génétique par construction de modèle probabiliste, et plus particulièrement par *linkage learning*. Cet algorithme infère des dépendances entre les variables du problème afin de mener une exploration plus pertinente. Cette approche générique présente une avance conséquente dans le domaine, par

sa capacité à résoudre des problèmes classiques inabordables par des approches traditionnelles (Gras, 2004).

2.6.2 L'algorithme génétique simple

L'algorithme génétique simple (AGS) est une métaheuristique itérative qui échantillonne simultanément plusieurs points de l'espace de recherche. Les algorithmes génétiques ont été inspirés des concepts biologiques de l'évolution. Cette origine leur confère une terminologie particulière. Ainsi, une itération est appelée *génération*, un point de l'espace de recherche est appelé un *chromosome*, l'ensemble des points considérés à une itération donnée est appelé *population*. Cette population est itérativement modifiée par trois opérateurs, chacun fonctionnant sur son propre voisinage : l'opérateur de *sélection*, l'opérateur de *recombinaison*, et finalement l'opérateur de *mutation*. Par un souci de clarté, nous associerons souvent à cette terminologie les termes plus génériques définis dans les Sections précédentes.

Le fonctionnement d'un AGS commence par une initialisation de la population. Un nombre donné Tp de solutions potentielles (points de l'espace de recherche correspondant donc chacun à une instanciation donnée des variables du problème) sont aléatoirement choisis dans X pour former la population initiale. Généralement, la taille de la population Tp se situe entre une centaine et quelques milliers. Ensuite, deux étapes se succèdent de façon itérative :

1. *La sélection.* Un nombre Tp de solutions sont choisis stochastiquement avec remise à partir de la population. Ce choix est effectué avec un biais pour les solutions dont la valeur de fitness est élevée, en regard du reste de la population. Certaines solutions pourront être sélectionnées plusieurs fois alors que d'autres seront éliminées du processus. Cette étape nécessite bien sûr l'évaluation préalable de toutes les solutions de la population par la fonction objectif.
2. *La reproduction.* Les solutions sélectionnées à l'étape (1) sont stochastiquement modifiées par deux opérateurs : la recombinaison et la mutation. Les résultats de ces opérations sont insérés dans une nouvelle population, et le processus recommence à l'étape (1).

L'opérateur de sélection dans un AGS est de type "roue de loterie". Cet opérateur prend en entrée l'ensemble des chromosomes de la population et en produit un seul. Il fonctionne en associant à chaque chromosome de la population une probabilité d'être sélectionné. Pour une population de chromosomes $c_1, c_2, \ldots, c_{Tp}$, la probabilité de sélectionner c_i est définie par :

$$p(c_i) = \frac{OF(c_i)}{\sum_{j=1}^{Tp} OF(c_j)}$$

Ainsi, plus la valeur de fitness d'une solution est grande par rapport aux autres et plus cette solution a de chance d'être sélectionnée. C'est cette opération qui est responsable de la convergence vers des solutions de valeur plus élevée. L'opérateur de mutation est unaire, il produit une solution à partir d'une solution. Son opération consiste à initialiser avec une probabilité donnée $pMut$ l'instanciation de chaque variables de la solution. Cet opérateur est appliqué sur toutes les solutions sélectionnées.
L'opérateur de recombinaison est binaire, il produit deux solutions à partir de deux solutions. Son fonctionnement consiste à échanger l'instanciation d'une succession de variables

Pseudocode 1 Algorithme Génétique Simple

choisir $max\_iteration$
choisir Tp
choisir $pRec$
choisir $pMut$
initialise(population(0))
$i = 0$
TANT QUE $i \leq max\_iteration$ {
  évalue(population(i))
  $j = 0$
  TANT QUE $j \leq Tp$ {
    $x_1 \leftarrow$ sélection(population(i))
    $x_2 \leftarrow$ sélection(population(i))
    SI tirage aléatoire biaisé selon $pRec$ réussi :
      $(x_1, x_2) \leftarrow$ recominaison(x_1, x_2)
    POUR TOUS $v_k \in x_1, x_2$
      SI tirage aléatoire biaisé selon $pMut$ réussi :
        $v_k \leftarrow$ mutation(v_k)
    insère x_1 et x_2 dans la population$(i + 1)$
    $j \leftarrow j + 2$
  }
  $i \leftarrow i + 1$
}

entre les deux solutions. La Figure 2.3 illustre le procédé en détail. L'opérateur de recombinaison est appliqué avec une probabilité $pRec$ sur les solutions sélectionnées. Comme nous avons pu le voir, seul l'opérateur de sélection choisit un élément du voisinage avec un biais pour les valeurs élevées. C'est donc cette opération qui permet à l'algorithme génétique de considérer des points de valeur de plus en plus haute et donc de converger. L'opérateur de recombinaison cherche à combiner des "briques élémentaires" dans le but de créer de nouvelles solutions prometteuses. L'opérateur de mutation sert à ajouter de la diversité. Un algorithme génétique fonctionne donc sur trois paysages distincts : le paysage de sélection, le paysage de recombinaison et le paysage de mutation. Le fonctionnement détaillé de l'AGS est décrit dans le Pseudocode 1.

De nombreuses variantes existent suivant les opérateurs utilisés. Citons notamment la sélection par tournoi, élitiste ou tronquée. Ces différentes formes permettent de faire varier la pression sélective exercée sur la population, c'est à dire l'importance du biais statistique vers les meilleures solutions (Blickle and Thiele, 1995). Différentes formes de recombinaison sont également couramment utilisées dont les plus classiques sont la recombinaison uniforme et à un ou deux points.

L'ajustement de plusieurs paramètres est nécessaire au fonctionnement d'un algorithme génétique : la taille de la population Tp, les probabilités des opérateurs génétiques $pMut$ et $pRec$, plus éventuellement d'autres paramètres concernant la pression sélective. De nombreuses études expérimentales ont été menées pour évaluer l'influence de ces paramètres. Il en ressort en général que le réglage optimal de ces paramètres est fortement dépendant du problème que l'on cherche à optimiser. D'autres études ont également été menées (Baluja,

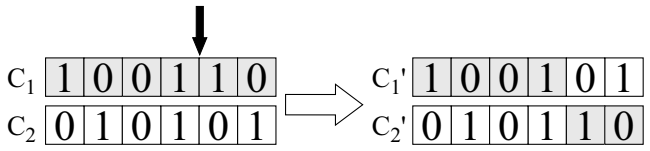


Fig. 2.3 : Illustration d'une recombinaison à un point. Les deux chromosomes (solutions potentielles) c_1 et c_2 sont considérés comme des chaînes de caractères, chaque symbole correspondant à l'instanciation (0 ou 1) de la variable concernée. Les deux chromosomes sont appariés l'un au dessous de l'autre et une position de recombinaison, symbolisée par la flèche noire, est choisie aléatoirement dans cet appariement. Tous les symboles situés après cette position sont échangés entre les deux chromosomes. Dans le cas présenté, il existe cinq positions d'échange possible. Le voisinage de l'opérateur est donc pour cet exemple composé de cinq couples de chromosomes. Cet opérateur consiste donc à choisir aléatoirement un élément de son voisinage.

1996; Goldberg and Segrest, 1987; Jones, 1995; Sharpe, 2000) pour comparer les capacités d'optimisation d'un algorithme génétique avec des approches de grimpeur strict à départs multiples. Les résultats montrent que dans la grande majorité des cas, des approches simples de grimpeur sont souvent plus efficaces qu'un algorithme génétique. Cette observation signifie que l'application directe d'un algorithme génétique à un problème est souvent une mauvaise idée. Un tel algorithme peut se révéler efficace, mais à la condition de définir des fonctions de voisinages et des opérateurs de déplacement adaptés, en fonction de toutes les connaissances préalables que l'on a du problème.

2.6.3 Les approches évolutionnistes par découverte de dépendances

Les approches par découverte de dépendances ("linkage learning"), ont été imaginées pour traiter les difficultés présentées dans la Section précédente. Nous nous intéresserons plus particulièrement à une sous-classe appelée *probabilistic linkage learning*.

L'algorithme BOA (Bayesian Optimization Algorithm) (Pelikan et al., 1999; Pelikan, 2002) est une méthode évolutionniste. Sa particularité réside dans le fait qu'elle n'utilise pas, comme les méthodes classiques, un ensemble fini de fonctions de voisinage et d'opérateurs, mais une modélisation probabiliste de la population, laquelle est considérée comme un échantillon de points de l'espace de recherche biaisés vers les valeurs hautes. Son fonctionnement de base se résume dans les étapes suivantes :

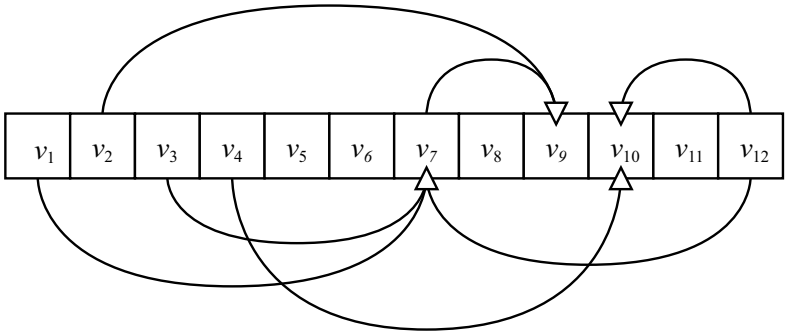
1. *Initialisation* : $t = 0$, La population initiale $P(t)$ est générée.
2. *Sélection* : Sélection de $S(t)$ à partir de $P(t)$ (sélection biaisée en faveur des meilleures solutions.
3. *Inférence d'un modèle* : On construit un modèle probabiliste sous la forme d'un réseau bayésien de dépendances entre les variables (Figure 2.4). La topologie du réseau ainsi que les valeurs de probabilité sont inférées.
4. *Génération* : On génère un nouvel ensemble de solutions $O(t)$ à partir du modèle inféré.
5. *Remplacement* : On crée une nouvelle population $P(t+1)$ en remplaçant certaines solutions de $P(t)$ par des solutions de $O(t)$, et on incrémente t.
6. *Terminaison* : Si le critère de terminaison n'est pas satisfait, aller en (2).

L'étape clé de la méthode est clairement celle où l'on infère le modèle bayésien. En analogie avec les méthodes classiques, le réseau bayésien peut être vu comme une fonction de voisinage. C'est d'ailleurs sur ce point que BOA présente une avance conséquente dans le domaine, car en inférant un réseau de dépendances, cette méthode modifie à chaque itération le voisinage qu'elle utilise en fonction de la spécificité propre du problème qu'elle optimise. Cette étape nécessite deux choses : découvrir la structure du réseau, c'est-à-dire découvrir les dépendances, et déterminer les probabilités conditionnelles associées aux variables. Déterminer les probabilités se fait très simplement à partir de la fréquence observée dans la population des occurrences simultanées de chaque instance de chaque variable. L'apprentissage du réseau est un problème beaucoup plus complexe puisqu'il s'agit de déterminer quel réseau parmi les $2^{(n^2)}$ possibles représente le mieux les données. Ce problème est NP-complet en soi (Chickering et al., 1997) et BOA utilise donc une heuristique non-exacte pour construire le réseau. La méthode utilisée est gloutonne et contruit le réseau petit à petit, à partir d'un réseau initial sans arc. Pour fonctionner, BOA a besoin du nombre maximum de dépendances k que peut avoir une variable. Il calcule ensuite à partir de cette valeur la taille de la population nécessaire pour assurer, avec un niveau d'erreur choisi, que toutes les instanciations possibles de tous les blocs de taille k soient présentes au moins une fois dans la population initiale. Cette contrainte limite à un dégré d'épistasie bas les problèmes à traiter, cependant cette méthode présente des avantages réels par rapport aux algorithmes génétiques sur des "benchmarks" d'optimisation comprenant des problèmes artificiels où le degré d'épistasie peut être modulé (Gras, 2005). Une extension à l'approche BOA appelée hBOA ("hierarchical bayesian optimization algorithm") (Pelikan and Goldberg, 2001) a également été proposée. Son but est de résoudre des problèmes trompeurs à plusieurs niveaux, c'est-à-dire des problèmes qui sont décomposables en une hiérarchie de sous-problèmes, chacun ayant sa propre sous-fonction objectif.

Bien que très puissantes, ces approches restent très limitées sur les applications réelles car le degré d'épistasie peut être trop important pour permettre une résolution par ces approches et une métaheuristique *spécifique* (approche experte) au problème sera dans la majorité des cas certainement préférable. Nous avons ainsi testé hBOA sans succès sur un problème d'alignement multiple, et bien qu'il soit capable de converger vers des sous-solutions pertinentes, il reste loin derrière les algorithmes spécifiques au problème.

2.7 Discussion

Il apparaît clairement que le type de voisinage utilisé est prépondérant pour résoudre un problème d'optimisation combinatoire donné. Le choix d'un grimpeur strict par rapport à un recuit simulé est beaucoup moins important que le choix ou la définition d'une fonction de voisinage, laquelle va déterminer un paysage. Un même problème peut être insoluble pour un voisinage donné ou très facile pour un autre, car le nombre de maximums locaux peut être réduit ainsi que le nombre moyen d'étapes nécessaires pour converger. Dans tous les cas, une réflexion sur les particularités du problème s'impose afin de tirer parti de toutes les connaissances dont on peut disposer. Les algorithmes par découverte de dépendances marquent une avance significative dans le domaine, mais ils sont limités par le nombre de dépendances qu'ils peuvent modéliser.



v_1	v_3	v_{12}	$P(v_7 = 0 \mid v_1, v_3, v_{12})$
0	0	0	p_1
0	0	1	p_2
0	1	0	p_3
...	...	...	...
1	1	1	p_{2^k}

Fig. 2.4 : Cette illustration représente un exemple de réseau bayésien que BOA pourrait inférer sur un problème à 12 variables. Les dépendances entre variables sont représentées par des arcs. Par exemple, v_7 dépend de v_1, v_3 et v_{12}. La topologie du réseau est inférée par une heuristique gloutonne non-exacte. Les probabilités conditionnelles associées aux variables sont estimées par simple comptage, comme représenté dans le tableau du bas : à chaque combinaison d'instanciations des variables dont dépend une variable donnée est associé une probabilité conditionnelle correspondant au nombre de fois où l'instanciation en question a été observée dans la population.

Chapitre 3

Etat de l'art des méthodes de découvertes de similarités dans les séquences de biopolymères

3.1 Introduction

Ce chapitre est destiné à donner un aperçu des principales méthodes impliquées dans la découverte de similarités dans un ensemble de séquences. Vu la diversité et le nombre de ces méthodes, nous ne serons pas exhaustifs. Notre souci sera plutôt d'établir une classification et de décrire les principales stratégies à travers les algorithmes les plus connus. Le lecteur intéressé pourra trouver des informations complémentaires dans les revues suivantes : (Brazma et al., 1998; Brejova et al., 2000; Notredame, 2002).

Les méthodes de découverte de similarités incluent les méthodes de découverte de modèles déterministes, souvent appelés *patterns*, et les méthodes d'alignement. Toutes ces méthodes recherchent des similarités présentes dans un ensemble de séquences protéiques ou nucléiques, supposées apparentées voir homologues. Il existe également des approches de *discrimination*, qui recherchent à partir d'un ensemble de séquences dit positif, les régularités n'étant pas ou peu présentes dans un deuxième ensemble dit négatif. Ce dernier type d'approche n'est pas traité dans cette présentation.
Les méthodes de découverte de patterns recherchent des similarités sous la forme d'une sous-classe des expressions régulières, dont les spécificités et définitions précises varient beaucoup selon la méthode. Un pattern a un nombre fini d'*occurrences* dans l'ensemble de séquences, correspondant aux sous-chaînes dans les séquences qui sont représentées par le pattern, soit exactement, soit dans des limites fixées (nombre d'erreurs autorisées).

Les méthodes d'*alignement* cherchent à représenter explicitement des relations d'homologie (supposées) entre les résidus des séquences. Elle supposent l'homologie à partir de similarités que les séquences peuvent présenter. Les méthodes dites *globales* forcent tous les résidus de l'ensemble de séquences à participer à l'alignement alors que les méthodes *locales* ne cherchent à aligner que les parties les plus significatives. Les alignements se représentent par une superposition des séquences ou de parties de séquences, en les arrangeant de façon à ce que les résidus supposés homologues soient *alignés* dans une même colonne. Lorsque plus de deux séquences sont alignées, on parle d'*alignement multiple*.

Les alignements multiples de séquences représentent un véritable challenge combinatoire. Ils permettent de conduire des études évolutionnaires sur une famille de séquences, ou de calculer un modèle dans le but d'identifier des séquences inconnues comme membres potentiels de la famille. La procédure d'alignement multiple est, tout comme l'alignement deux à deux, intensivement utilisée en bioinformatique et en biologie moléculaire. L'alignement multiple de séquences pose en plus du défi combinatoire, le problème de la définition d'une fonction objectif qui doit être capable d'assigner une valeur de "signifiance biologique" à un alignement donné.
On peut globalement discerner deux types d'approches. L'approche la plus courante concerne les techniques basées sur des algorithmes de programmation dynamique, et cherche généralement à effectuer des alignements multiples globaux, où les séquences sont alignées sur toute leur longueur. La grande majorité d'entre elles déterminent l'alignement multiple par une stratégie gloutonne sur la base d'une information obtenue par une comparaison *deux à deux* des séquences. L'autre approche regroupe des méthodes plus spécifiques qui effectuent des alignements locaux multiples tout en calculant les paramètres d'un *modèle probabiliste* de l'alignement.
Ces approches utilisent généralement une optimisation par voisinage, qui considère toutes les séquences *simultanément*, permettant ainsi de repérer des homologies plus faibles, qui ne sont pas détectables par une comparaison deux à deux. La majeure partie de ces approches est basée sur des algorithmes statistiques d'optimisation, comme l'algorithme "expectation-maximization" (EM), et l'échantillonnage de Gibbs (Gelman et al., 2004). Les optimisations du modèle probabiliste et de l'alignement sont intimement liées. On peut en effet toujours générer un alignement à partir d'un modèle probabiliste voir même d'un pattern en alignant directement les occurrences du modèle. On peut réciproquement déterminer les paramètres d'un modèle par calcul direct à partir d'un alignement.

Nous utiliserons dans les descriptions qui vont suivre ces notations de base. L'ensemble de séquences considéré est donné par $S = \{s_1, s_2, \ldots, s_T\}$ où s_i est la i^{eme} séquence de S. L'ensemble est constitué de T séquences, toutes supposées de la même longueur L. Ces séquences sont définies sur un alphabet Σ de cardinal K.

La Section 3.2 traitera des méthodes de découverte de patterns. La Section 3.3 décrira l'approche permettant l'alignement de deux séquences par programmation dynamique, approche qui est à la base de la grande majorité des méthodes d'alignements multiples. La Section 3.4 présente la problématique de l'alignement multiple. Nous y aborderons les approches classiques ainsi que les principaux scores qui leur sont associés.

3.2 Découverte de patterns

3.2.1 Généralités

Les méthodes de découverte de patterns déterminent à partir de S un ou plusieurs modèles déterministes, se présentant sous la forme d'une sous-classe des expressions régulières. Une *occurrence* d'un pattern dans S est un mot de S, pouvant être représenté par le pattern, d'une façon exacte ou dans des limites fixées. Le nombre minimum d'occurrences que doit avoir un pattern est appelé le *support*. Les spécificités des patterns inférés par les différentes méthodes sont assez variables, mais généralement ces spécificités peuvent prendre en considération une ou plusieurs des propriétés suivantes :

- *Le symbole joker* est un symbole particulier qui permet de représenter n'importe quel symbole de Σ. Il est généralement noté par '.' ou 'x'.

- *Les régions flexibles* permettent de définir dans un pattern une ou plusieurs positions pour lesquelles on autorise un nombre variable de symboles joker. Une région flexible est généralement spécifiée par le nombre minimum et maximum de symboles joker autorisés. Elles sont généralement notées $\mathrm{x}(i, j)$ pour signifier au moins i et au plus j symboles joker.

- *Les symboles ambigus* sont des symboles appartenant à un alphabet *étendu*. Ils permettent de représenter directement un sous ensemble de Σ. Ils sont particulièrement utiles dans le cas des protéines pour lesquelles certains acides aminés ont des propriétés communes. Il est pertinent dans ce cas de définir des symboles ambigus correspondant à un sous ensemble de Σ. Ces sous-ensembles sont généralement définis explicitement entre crochets. Par exemple [ILV] est un symbole ambigu représentant soit 'I' soit 'L' soit 'V'. Il existe également un alphabet étendu standard (IUPAC-IUB) pour les nucléotides (Cornish-Bowden, 1985) (Table 3.1), qui définit un code à une lettre pour toutes les ambiguïtés possibles.

Code	Description	nucléotides
G	Guanine	
A	Adenine	
T	Thymine	
C	Cytosine	
R	Purine	(A or G)
Y	Pyrimidine	(C or T or U)
M	Amino	(A or C)
K	Ketone	(G or T)
S	Strong interaction	(C or G)
W	Weak interaction	(A or T)
H	Not-G	(A or C or T) H follows G in the alphabet
B	Not-A	(C or G or T) B follows A
V	Not-T (not-U)	(A or C or G) V follows U
D	Not-C	(A or G or T) D follows C
N	Any	(A or C or G or T)

Tab. 3.1 : Code IUPAC-IUB une lettre pour l'ADN.

Un exemple de pattern incluant toutes ces propriétés est : [ab] − x − [a] − [bcd] − $\mathrm{x}(3, 5)$ − [e]. Cette notation indique que la première position peut correspondre aux symboles 'a' ou 'b', suivis de n'importe quel symbole, puis un 'a', suivi d'un 'b', 'c' ou d'un 'd', puis un nombre de joker compris entre 3 et 5, suivis d' un 'e'. La plupart des méthodes recherchent des patterns dits *maximaux* ce qui signifie que l'on ne peut pas augmenter leurs spécificités sans diminuer leurs supports. Certaines méthodes infèrent des patterns qui doivent avoir un nombre minimum d'occurrences *exactes* dans S alors que d'autres autorisent des occurrences *approchées* dans les limites d'une tolérance fixée.

3.2.2 Recherche exhaustive

La recherche exhaustive est l'approche la plus simple. Elle consiste à énumérer tous les patterns possibles d'une classe donnée afin d'identifier tous ceux qui satisfont les contraintes requises. Cette approche étant exponentielle sur la longueur du pattern, elle n'est utilisable que pour des patterns simples (sans régions flexibles) et définis sur un cardinal bas. Quasiment toutes les méthodes de recherche exhaustive sont limitées à des ensembles de séquences d'ADN ($K = 4$). Ce type d'approche garantit par contre de reporter tous les patterns satisfaisant les contraintes requises. L'algorithme "oligo-analysis" (van Helden et al., 1998) énumère tous les oligonucléotides d'une longueur inférieure à 10 et reporte ceux qui sont statistiquement considérés comme sur-représentés par rapport à ce que l'on attendrait par hasard. L'algorithme "YMF" (Yeast Motif Finder) (Sinha and Tompa, 2002) est spécifique pour des promoteurs de levures et permet d'inférer des patterns courts avec éventuellement une région variable au milieu. MOTIF (Smith et al., 1990), est un algorithme permettant d'inférer des patterns simples sur des séquences protéiques. Étant donné que le cardinal des protéines ($K = 20$) est beaucoup plus élevé que celui de l'ADN, les patterns que peut produire cette méthode sont extrêmement contraints. Ces patterns peuvent représenter trois acides aminés séparés par deux régions fixes composées d'un nombre donné de jokers, par exemple : 'A...Q....I'. La longueur maximale que peut prendre les régions jokers est spécifiée par l'utilisateur. Si les régions variables peuvent contenir au plus d symboles jokers, la combinatoire sur le nombre de patterns possibles sera de : $20^3 d^2$. L'algorithme aligne ensuite les occurrences de ces patterns pour essayer de les étendre en patterns plus spécifiques.

3.2.3 Recherche par profondeur dans un arbre

La recherche exhaustive devient rapidement impossible pour chercher des patterns plus compliqués, permettant des symboles ambigus et des régions variables. Une alternative consiste à construire les patterns, en partant d'un seul symbole, puis chercher à les étendre de toutes les façons possibles. Si une extension produit un pattern qui ne satisfait plus la contrainte de support, elle est abandonnée. Ce type d'approche est une recherche par profondeur dans l'arbre de tous les patterns possibles. Chaque extension correspond à un noeud dans l'arbre, et si elle est abandonnée, le sous-arbre correspondant est élagué. Bien que cette approche ait toujours une complexité exponentielle dans le pire des cas, elle permet en fonction de la qualité de l'élagage, de produire des patterns d'un niveau de représentation supérieur à ce qui peut être fait par une recherche exhaustive. Cette approche est utilisée par l'algorithme Pratt2 (Jonassen, 1997), un algorithme approché, mais qui peut garantir l'exactitude du résultat sous certaines contraintes. Ce programme a été développé à la suite d'une première version, Pratt (Jonassen et al., 1995), qui avait le défaut de produire de grosses quantités de patterns non maximaux (patterns déjà contenus dans un pattern plus spécifique).
Pratt2 permet de découvrir des patterns définis sur un alphabet étendu et comportant des régions flexibles. Cet algorithme requiert plusieurs paramètres, définissant les spécificités des patterns qu'il recherche. L'utilisateur doit spécifier le support k, correspondant au nombre minimum de séquences qui doivent avoir au moins une occurrence de chaque pattern produit. Un ensemble de paramètres et de bornes définissent le type de pattern recherché : $\mathcal{B} = (\mathcal{A}, \mathcal{P}, \mathcal{L}, \mathcal{W}, \mathcal{F}, \mathcal{N}, \mathcal{FP})$. $\mathcal{A}$ est un alphabet étendu de Σ, sur lequel les patterns sont définis. $\mathcal{P}$ est le nombre maximum de symboles contenus dans les patterns, $\mathcal{L}$ est la longueur maximum du pattern, laquelle correspond à la somme du nombre de

symboles et du nombre de régions flexibles, $\mathcal{W}$ est la longueur maximum des régions flexibles, avec j pour la longueur d'une région flexible $x(i,j)$. $\mathcal{F}$ est la flexibilité maximum des régions flexibles, avec $j-i$ pour la flexibilité de $x(i,j)$. $\mathcal{N}$ est le nombre maximum de régions flexibles et $\mathcal{FP}$ est la valeur maximum que le produit de toutes les flexibilités peuvent avoir (ce produit est défini par $\prod_k(i_k - j_k + 1)$). La méthode ne garantit pas de trouver tous les patterns satisfaisant les contraintes (heuristique approchée), sauf sous la condition d'interdire les régions flexibles. Le type de patterns produits est de la forme : $A_1 - x(i_1,j_1) - A_2 - x(i_2,j_2) \ldots - x(i_n,j_n) - A_N$. $A_1 \ldots A_n$ sont les symboles du pattern défini sur $\mathcal{A}$ et $x(i,j)$ représente un nombre de i à j symboles jokers. $i-j$ définissant la flexibilité de la région. Un exemple de pattern pourrait être $A-[DE]-x(3)-G-x(3,4)-L$.

L'algorithme commence la recherche avec un pattern composé d'un seul symbole, puis il cherche à l'étendre de toutes les façons possibles, définies par les contraintes $\mathcal{B}$. L'heuristique de Pratt repose sur un score qui permet d'estimer la spécificité maximum que pourra atteindre un pattern en cours d'extension. Si un pattern A a au moins les mêmes occurrences qu'un autre pattern B plus spécifique déjà trouvé, et que le score maximum estimé de A est inférieur à celui de B, il est abandonné. Cette heuristique est exacte dans le cas où les régions variables ne sont pas autorisées.

3.2.4 Recherche de patterns guidée par l'ensemble de séquences

Une autre approche consiste à utiliser S pour guider la construction des patterns. Les deux algorithmes les plus connus utilisant cette approche sont TEIRESIAS (Rigoutsos and Floratos, 1998) et Splash (Califano, 2000). Ces deux algorithmes recherchent des patterns qui satisfont une contrainte de *densité*, laquelle spécifie le ratio minimum de symbole non joker sur une longueur donnée. Ces deux algorithmes partagent beaucoup de similarités. Une différence importante cependant concerne le fait que Splash peut trouver des patterns définis sur un alphabet étendu alors que TEIRESIAS est limité à Σ et au symbole joker. Nous ne décrirons que l'algorithme le plus récent.
Splash est un algorithme exact qui permet de produire sous certaines contraintes tous les patterns maximaux qui sont communs à au moins deux séquences de S. Les patterns produits sont de la forme $(\Pi \cup {'.'})^*$ où Π est l'alphabet sur lequel les patterns sont définis et $'.'$ représente le symbole joker. Un symbole de Π est appelé un *symbole plein*. Une contrainte de densité doit être spécifiée par deux valeurs k_0 et l_0, pour limiter la recherche. Ces valeurs précisent le nombre k_0 minimum de symboles pleins dans un segment contigu de longueur l_0 du pattern et ne commençant pas par un joker. Par exemple, le pattern $A....BC....D$ satisfait la contrainte de densité : $k_0 = 3, l_0 = 7$ car les deux sous-chaînes $A....BC$ et $BC....D$ de longueur 7 et commençant par un symbole plein contiennent au moins trois symboles pleins. L'utilisateur spécifie en plus le support j_0 correspondant au nombre minimum d'occurrences qu'un pattern doit avoir dans l'ensemble de séquences, ainsi que m_0, le nombre minimum de symboles pleins que chaque pattern produit doit contenir. Ces deux dernières valeurs sont utiles pour éviter que le programme ne produise des patterns non significatifs.

L'alphabet Π peut être équivalent à Σ et dans ce cas les patterns produits sont appelés *patterns identiques*. On peut cependant aussi définir Π de façon à ce que chacun de ses symboles représente un ou un groupe de symboles de Σ. On parle dans ce cas de *patterns approximatifs* lesquels peuvent produire des occurrences non identiques, mais similaires dans l'ensemble de séquences. Dans tous les cas, les cardinaux de Π et Σ doivent être

identiques et Π_i, le i^{eme} symbole de Π doit représenter au moins Σ_i, le i^{eme} symbole de Σ. Le programme propose une option pour générer un tel alphabet étendu à partir d'une matrice de substitution BLOSUM (Henikoff and Henikoff, 1992).

Le fonctionnement de Splash repose sur une série d'opérateurs appliqués récursivement sur un ensemble de patterns de départ appelés *graines*, afin de les étendre en patterns maximaux satisfaisant les contraintes précitées. L'ensemble de graines correspond à tous les patterns de longueur inférieure ou égale à l_0 satisfaisant la contrainte de densité et de support. Cet ensemble est généré en énumérant tous les patterns de longueur l_0 ou inférieure présents dans S. Cet étape conduit à un ensemble de graines, associées avec les positions de leurs occurrences dans S. L'algorithme cherche ensuite à étendre chaque graine par l'application récursive de trois opérateurs ayant pour but d'augmenter la taille des pattern initiaux ou leur nombre de symboles pleins. Ces opérateurs fonctionnent en se référant aux occurrences du pattern dans S. Ils regardent sur les occurrences du pattern si un symbole commun peut être rajouté sans diminuer le support. Si un pattern ne peut plus être étendu d'aucune manière, il est maximal et est produit en résultat.

Bien qu'aucune complexité ne soient directement mentionnée, les auteurs donnent des valeurs de temps CPU sur divers ensembles de séquences et diverses contraintes. L'algorithme est capable de trouver tous les patterns maximaux (environ 600) de contrainte de densité $k_0 = 2, l_0 = 5$ présents dans au moins 20% des séquences d'une base de donnée randomisée de 512000 acides aminés en moins de 10 secondes, et en utilisant une quantité négligeable de mémoire.

3.2.5 Recherche de patterns présentant des occurrences avec erreurs

Certains algorithmes (fonctionnant principalement sur des séquences d'ADN) recherchent un pattern d'une longueur l donnée et défini sur Σ uniquement, pattern qui a un nombre minimum d'occurrences dans S avec exactement k ou au plus k erreurs. La particularité de ce type de problème est que le pattern recherché n'a pas forcément d'occurrences exactes dans S mais un ensemble d'occurrences approchées, qui peuvent être très différentes les unes des autres lorsqu'on les compare deux à deux.
Cette problématique s'est considérablement développée à la suite du lancement d'un problème challenge (Pevzner and Sze, 2000), lequel est décrit et commenté en détail dans la Section 5.2. Prenons en exemple le paramétrage de base du problème challenge : on commence par générer aléatoirement $T = 20$ séquences sur un alphabet de cardinal $K = 4$ (ADN) de longueur 600. On génère ensuite un pattern de longueur $l = 15$ (sur l'alphabet Σ) que l'on insère exactement une fois par séquence, mais en effectuant à chaque insertion $d = 4$ substitutions aléatoires. Deux insertions peuvent donc avoir jusqu'à huit symboles différents. Énumérer tous les patterns de longueur 15 et vérifier s'ils ont une occurrence par séquence avec quatre erreurs n'est clairement pas réalisable en un temps raisonnable. Un certain nombre d'algorithmes traitant cette problématique existent, dont la plupart sont dédiés au challenge. Citons notamment WINNOVER (Pevzner and Sze, 2000), PROJECTION (Buhler and Tompa, 2001; Buhler and Tompa, 2002), en particulier PatternBranching (Price et al., 2003), Multiprofiler (Keich and Pevzner, 2002), et SMILE (Marsan and Sagot, 2000; Marsan, 2002). Il faut préciser que SMILE n'est pas dédié au challenge, mais présente des fonctionnalités plus générales. Il peut trouver un pattern à une distance de Hamming d'au plus d de ses occurrences, et peut en fonction du choix de l'uti-

lisateur autoriser une ou plusieurs régions variables dans le pattern. Il permet également de rechercher des patterns sur un alphabet étendu.

WINNOVER propose de travailler directement à partir des facteurs de S de longueur 15. Cette approche construit un graphe où chaque noeud est un facteur de longueur 15 et tous les facteurs sont représentés. Deux noeuds sont reliés par un arc s'ils n'appartiennent pas à la même séquence et s'ils ont un nombre de symboles différents plus petit ou égal à $2d$. Un graphe $T-$parti[1] ($T = 20$) est ainsi obtenu dans lequel les 20 occurrences du signal forment une clique de taille 20. La recherche de cliques dans un graphe est un problème NP-difficile. Le fonctionnement de Winnover est basé sur l'idée de réduire la taille du graphe en supprimant les noeuds dont on est sûr qu'ils n'appartiennent pas à une grande clique. Le programme utilise pour cela la notion de clique augmentable. Un noeud u est voisin de la clique $C = \{v_1, v_2, \ldots, v_k\}$ si $\{v_1, v_2, \ldots, v_k, u\}$ forme également une clique, et une clique est dite *augmentable* si elle a au moins un voisin dans chacune des T parties du graphe. Le principe de réduction du graphe est basé sur l'observation que tous les arcs d'une clique maximale de taille T font également partie d'au moins $\binom{T-2}{k-2}$ cliques augmentables de taille k. Le programme retire alors tous les arcs qui appartiennent à moins de $\binom{T-2}{k-2}$ cliques. Le procédé est itéré jusqu'à l'obtention d'une petite collection de cliques augmentables dans lesquelles la clique maximale de taille T aura une chance d'être trouvée en un temps raisonnable.

L'algorithme PROJECTION est une approche probabiliste, dont l'idée principale consiste à utiliser des fonctions de hachage. Le principe consiste à transformer un mot de longueur l en un autre mot de longueur d formé par la concaténation des symboles choisis dans d positions parmi l. Une fonction donnée résulte du choix aléatoire des d positions qui sont concaténées. Tous les facteurs de S de longueur l sont transformés par une fonction de hachage donnée. Si cette fonction produit un mot plus souvent que les autres, il y a de bonne chance qu'il soit issu d'une partie non négligeable des occurrences du mot inséré.

L'algorithme PatternBranching consiste en un grimpeur strict sur le voisinage de Hamming. Cet algorithme est le plus efficace pour résoudre le problème challenge, même sur des instances plus difficiles du problème. L'espace de recherche correspond à tous les mots de longueur l. Le grimpeur cherche à minimiser une fonction objectif correspondant à une somme de distances de Hamming minimales. Soit A un mot de longueur l et soit $H(A, P)$ la distance de Hamming entre A et P, un facteur de S. Soit $d(A, s_i) = \min(H(A, P), \forall P \mid P \in s_i)$. La fonction objectif est alors définie comme :

$$OF = \sum_{i=1}^{T} d(A, s_i)$$

soit la somme des distances de Hamming minimales pour chaque séquence entre A et tous les facteurs d'une séquence. Le grimpeur est lancé sur tous les facteurs de S et cela suffit généralement à identifier les occurrences du motif inséré.

Il faut noter que d'une façon générale, ce type de modèle n'est pas très significatif du point de vue biologique. En effet, ce modèle suppose que les substitutions sont

[1] Un graphe T-parti est également appelé $T-$colorable.

équiprobables pour toutes les positions. De plus dans le cas le plus contraint (chercher un mot qui a une occurrence par séquence avec *exactement* d erreurs), il suppose que toutes les occurrences du motif biologique recherché ont subit le même nombre de substitutions à partir d'un ancêtre commun, ce qui n'est que très rarement le cas dans la pratique.
Si l'on autorise *au plus* d erreurs, les occurrences d'un motif auront une chance d'être retrouvées pour autant que d soit petit par rapport à l. Dans le cas contraire, il est fort probable que ce type de modèle identifie un grand nombre d'occurrences, parmi lesquelles les occurrences réelles du motif ne seront pas discernables des occurrences non significatives.

3.3 Les alignements deux à deux par programmation dynamique

La publication décrivant l'algorithme de Needleman et Wunsch de programmation dynamique pour l'alignement de deux séquences (Needleman and Wunsch, 1970) est certainement l'une des les plus connues en bioinformatique. Cet algorithme permet d'effectuer un alignement global entre deux séquences de longueur n et m de façon optimale avec une complexité spatiale et temporelle de $\mathcal{O}(nm)$. Il existe plusieurs variantes, toutes basées sur la programmation dynamique (Gotoh, 1982; Smith and Waterman, 1981). Nous ne présenterons ici que l'algorithme de Needleman and Wunsh pour l'alignement global. Le lecteur intéressé pourra se référer au deuxième chapitre de (Durbin et al., 1998) dans lequel plusieurs de ces algorithmes sont décrits en détail.

Les algorithmes d'alignement de séquences par programmation dynamique sont intensivement utilisés en bioinformatique. Il sont de plus à la base de la majeure partie des programmes d'alignement multiples. L'alignement de deux séquences consiste à mettre en relation les similarités qu'elles peuvent présenter. Afin d'illustrer la nature d'un alignement, prenons un exemple d'alignement global entre deux protéines homologues de serpents : une neurotoxine du Bongare rayé (Bungarus multicinctus, NXLH5_BUNMU) et une cardiotoxine du Cobra cracheur (Naja sputatrix, CTX1_NAJSP) :

```
CTX1_NAJSP    MKTLLLTLVVVTIVCLDLGYTLKCNKLVPLFY-----KTCPAGKNLCYKIF
NXLH5_BUNMU   METLLLTLVVVTIVCLDLGYTMQCKTC--SFYTCPNSETCPDGKNICVKRS

CTX1_NAJSP    MVAT----PKVPVKRGCIDVCPKSSLLVKYVCCNTDRCN-
NXLH5_BUNMU   WTAVRGDGPKREIRRECAATCPPSKLGLTVFCCTTDNCNH
```

Deux symboles disposés l'un au-dessous de l'autre sont dits *appariés*. Certains appariements sont constitués de deux symboles identiques alors que d'autres concernent des symboles différents : un acide aminé ayant été substitué par l'autre au cours de l'évolution. Nous verrons plus loin que certaines substitutions sont biologiquement pertinentes car elle concernent des acides aminés aux propriétés physico-chimiques similaires, et sont plus souvent observées dans des alignements supposés "corrects" par rapport à ce que l'on attendrait par hasard. Un appariement de deux symboles suppose qu'ils sont homologues, c'est-à-dire qu'ils sont supposés descendre d'un ancêtre commun. Un alignement, est une description explicite des homologies supposées, contrairement aux patterns ou aux modèles probabilistes (que nous verrons plus loin). Un alignement suppose deux types de mutation : la *substitution* (un symbole est remplacé par un autre) et l'*insertion* ou la *délétion*.

Ce deuxième type de mutation est communément désigné par la contraction *indel* (pour INsertionDELetion) et est représenté par le symbole '-' dans l'alignement. Un indel correspond soit à l'insertion d'un ou de plusieurs symboles dans une séquence soit à une délétion de ces symboles dans l'autre séquence. On considère généralement la délétion ou l'insertion de plusieurs symboles contigus comme un seul événement mutationnel. Un alignement doit conserver l'ordonnancement des symboles dans les séquences. Des évènements évolutionnaires comme les réarangements ou les duplications ne sont donc pas modélisables par ce type de procédure.

3.3.1 Le système de score pour l'alignement de protéines

Établir un alignement entre deux séquences consiste à déterminer les appariements des symboles de façon à maximiser un système de score. Le score total d'un alignement correspond à la somme des scores attribués à chaque événement évolutionnaire constaté. Ce système additif suppose que les symboles sont indépendants les uns des autres.
Le premier type d'événement est la substitution d'un acide aminé par un autre. On a donc besoin d'un système de score qui attribue une valeur à tous les appariements possibles. Ces valeurs d'appariements sont données par des matrices dites de *substitution* comme les matrices BLOSUM (Henikoff and Henikoff, 1992), PAM (Dayhoff et al., 1978) et GONNET (Gonnet et al., 1992). Ces scores correspondent au ratio de la probabilité d'observer un appariement donnée dans un "vrai" alignement sur la probabilité d'observer cet appariement par hasard. Les matrices BLOSUM par exemple utilisent la base de données BLOCKS (Henikoff and Henikoff, 1991; Henikoff et al., 2000) d'alignements multiples locaux ("blocs") comme source d'appariements supposés corrects. Une matrice BLOSUM62 est construite à partir de l'ensemble des blocs présentant une similarité d'au moins 62% d'identité. Le principe est le suivant. Un décompte de tous les couples possibles d'acides aminés appariés (situés dans la même colonne d'un bloc) est effectué. Il y a $K(K+1)/2 = 210$ couples différents possibles. On obtient alors pour chaque couple d'acides aminés i et j une fréquence observée $q_{i,j}$ d'appariement (supposés biologiquement corrects). Ensuite la probabilité d'occurrence de l'acide aminé i dans un couple (i,j) (probabilité marginale) est donnée par :

$$p_i = q_{i,i} + \sum_{i \neq j} \frac{q_{i,j}}{2}$$

Les probabilités des couples attendus par hasard $e_{i,j}$ sont calculées par :

$$e_{i,j} = \begin{cases} (p_i)^2 & \text{si} \quad i = j \\ 2p_i p_j & \text{si} \quad i \neq j \end{cases}$$

Finalement, les valeurs de substitution de deux acides aminés dans la matrice BLOSUM sont données par un logarithme du ratio de q sur e arrondi à l'entier le plus proche :

$$B(i,j) = 2\log_2\left(\frac{q_{i,j}}{e_{i,j}}\right)$$

La matrice BLOSUM62 est présentée dans la Figure 3.1. On peut constater que les valeurs de substitution d'un acide aminé par lui-même ne sont pas les mêmes pour tous. Ainsi, le tryptophane ('W') obtient une valeur contre lui-même largement supérieure à celle des autres car sa substitution par un autre acide aminé est rarement observée. Les appariements dits *conservés* ont un score positif, ce qui signifie qu'on les observe plus souvent

dans des alignements "corrects" que ce que l'on attendrait par hasard. L'utilisation du logarithme permet d'obtenir un système de score additif.

Il faut également attribuer un score aux événements de type indels. On leur attribue un score devant être minimisé, on parle alors de coût d'indels. Un coût linéaire sur la longueur de l'indel $indel(g)$ avec g pour la longueur suppose un événement mutationnel distinct pour chaque symbole apparié avec un '-' :

$$indel(g) = g \cdot d$$

avec d étant le coût d'indel. Afin de considérer un succession d'indels comme un seul événement mutationnel, il est plus courant d'utiliser une fonction affine ou logarithmique sur g. Le coût pour une fonction affine est :

$$indel(g) = d + (g - 1)e$$

ou d correspond dans ce cas au coût d'ouverture d'un indel et e au coût de chaque symbole '-' contigu rajouté. Il n'existe pas de théorie pour leur attribuer une valeur comme c'est le cas pour les scores de substitutions (Vingron and Waterman, 1994) et leur ajustement dépend beaucoup des séquences que l'on aligne. Leur ajustement est alors empirique. Une technique est de les ajuster par rapport à la valeur la plus basse observée dans la matrice de substitution utilisée. Des valeurs typiques pour une fonction affine sont $d = 12$ et $e = 2$.

Le score d'un alignement correspond donc à une somme d'événements mutationnels entre les deux séquences. D'un point de vue statistique, ce score correspond au logarithme d'un ratio de vraisemblances : la vraisemblance que les deux séquences sont apparentées sur la vraisemblance qu'elles ne le sont pas.

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

Fig. 3.1 : La matrice de substitution BLOSUM62 est calculée à partir d'alignements locaux sans indels supposés corrects et présentant une similarité d'au moins 62% d'identité. Les valeurs sont des logarithmes du ratio des vraisemblances : fréquences observées d'un couple d'acide aminé donné sur la fréquence attendue par hasard.

3.3.2 L'algorithme de Needleman&Wunch

L'algorithme de Needleman&Wunch permet de déterminer l'alignement ou les alignements globaux de score optimal entre deux séquences. Le but informel étant d'effectuer le maximum d'appariements de score positif (appariements conservés) tout en minimisant le coût des appariements de score négatif, indels inclus. Nous présenterons l'algorithme sous sa forme la plus simple, en utilisant un système de score linéaire pour les indels. Supposons deux séquences x et y de longueur l avec x_i (resp. y_i) étant le i^{eme} symbole de x (res. y). Le score d'appariement entre deux symboles x_i et y_j est donné par la matrice de substitution $B(x_i, y_j)$ et le coût d'appariement d'un symbole avec un indel vaut d. L'idée est de construire une matrice N de taille $(l+1)(l+1)$ indexée par les deux séquences. Cette matrice est initialisée par $N(0,0) = 0$ et $N(0,i) = N(i,0) = i \cdot d$. Les valeurs restantes sont attribués par la relation récursive :

$$N(i,j) = \max \begin{cases} N(i-1, j-1) + B(x_i, y_j) \\ N(i-1, j) - d \\ N(i, j-1) - d \end{cases} \tag{3.1}$$

avec $N(i,j)$ donnant le score du meilleur alignement des deux préfixes se terminant respectivement aux positions x_i et y_j. On attribue donc à $N(i,j)$ le score qui maximise l'élongation de trois sous-alignements optimaux déjà calculés impliquant les préfixes se terminant par : x_{i-1} et y_{j-1} pour le premier, x_i et y_{j-1} pour le deuxième, et finalement x_{i-1} et y_j pour le troisième. La matrice est progressivement remplie depuis la case $N(1,1)$ jusqu'à la case $N(l,l)$, la valeur de cette dernière correspondra au score de l'alignement final. Pour obtenir l'alignement lui-même, il faut mémoriser à chaque étape du remplissage la provenance qui a maximisé l'équation 3.1. L'alignement correspondra alors à un chemin dans la matrice N. Le principe est illustré dans la Figure 3.2, qui représente la matrice pour deux séquences fictives. Les scores d'appariement sont issus de la matrice BLOSUM62, et un coût d'indel linéaire de $d = 3$.

Comme nous l'avons déjà mentionné, l'algorithme de N&W effectue un alignement sur toute la longueur des deux séquences. Cet algorithme ne produira donc un résultat satisfaisant que sur des séquences similaires sur toutes leurs longueur et de longueurs proches. Il est cependant fréquent de chercher à aligner des séquences distantes qui ne partagent plus que des similarités locales. Il existe pour cela l'algorithme de Smith&Waterman (Smith and Waterman, 1981) qui fonctionne en réinitialisant le score à zéro dès qu'il devient négatif. Il peut de cette manière mettre en évidence des similarités courtes (locales) entre deux séquences.

Ces algorithmes de programmation dynamique peuvent être généralisés pour l'alignement de séquences composées d'un nombre fini d'objets. Nous avons présenté le cas où un objet est un symbole, mais on peut étendre le principe à un séquence de n'importe quel type d'objets, pour autant que l'on définisse un système de score permettant d'attribuer une valeur à tous les appariements possibles de deux objets, ainsi que les scores de pénalités pour les indels. Un objet peut par exemple être un modèle probabiliste, qui décrit la probabilité de chaque symbole à une position donnée dans la séquence. On peut également, comme nous le verrons plus loin, aligner deux alignements selon ce principe.

L'utilisation la plus courante des algorithmes d'alignement deux à deux consiste à rechercher dans une base de donnée une séquence similaire à une séquence "requête". La

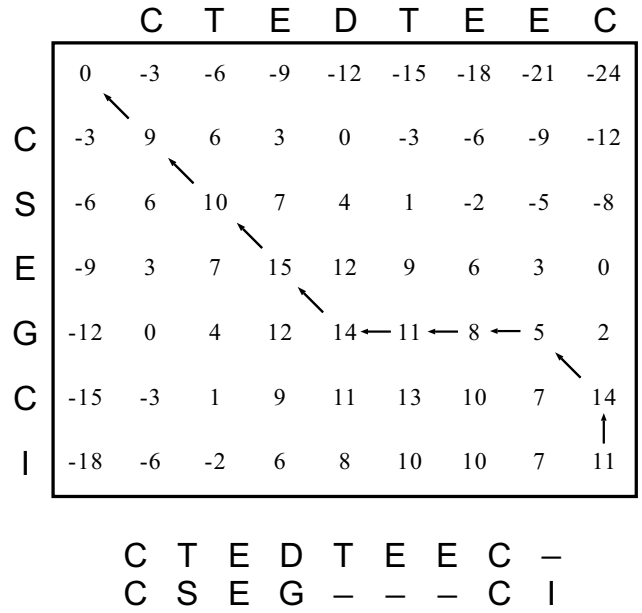


Fig. 3.2 : L'alignement de deux séquences fictives est présenté. Les scores d'appariement correspondent à la matrice BLOSUM62, et le coût d'indel est linéaire avec $d = 3$. Les flèches dans la matrice indiquent quelle provenance à été utilisée pour le calcul des cases qui correspondent à l'alignement optimal. Le chemin formé par les flèches permet de reconstruire cet alignement, lequel est présenté sous la matrice. Il n'existe pour cet exemple qu'un seul alignement optimal.

séquence requête est alignée avec toutes les séquences de la base de donnée et les séquences significativement similaires peuvent ainsi être identifiées. BLAST (Altschul et al., 1990) consiste en une plateforme de programmes d'alignement permettant d'effectuer des alignements d'une séquence requête contre toutes les séquences d'une base de données (ADN et protéines). Des alignements entre séquences protéiques et nucléiques peuvent également être obtenus en traduisant la séquence nucléique selon le code génétique et pour ses six cadres de lecture. L'algorithme de programmation dynamique utilisé est une heuristique non exacte permettant d'accélérer significativement le processus d'alignement afin de pouvoir l'effectuer contre des bases de donnée de centaines de milliers d'entrées. BLAST est de loin le programme le plus utilisé en bioinformatique.

3.4 l'Alignement multiple de séquences

Dès que l'on cherche à aligner plus de deux séquences, on parle d'alignement multiple. On peut globalement discerner deux classes de méthodes pour l'alignement multiple. Premièrement les méthodes basées sur la programmation dynamique, lesquels produisent des alignements multiples autorisant les indels. Ces alignements peuvent être globaux, quand l'on cherche à aligner tous les résidus de S, ou locaux si l'on n'aligne que les parties jugées significatives. Quasiment toutes les techniques de cette classe construisent l'alignement sur la base d'une information obtenue par comparaison deux à deux des séquences. La deuxième classe de méthode produit des alignements locaux multiples et sans indels, également appelés *blocs*. Ces alignements sont généralement obtenus par un processus

d'optimisation sur l'espace des alignements possibles. Contrairement au premier type, ils considèrent les séquences simultanément, permettant ainsi de détecter des similarités plus faibles. Il sont cependant plus spécifiques et donc plus contraints. Le travail effectué dans cette thèse concerne ce type d'alignements.

3.4.1 Généralisation de la programmation dynamique à des espaces multidimensionnels

Une première approche consiste à étendre à un espace multidimensionnel le principe de programmation dynamique présenté auparavant pour deux séquences. Le système de score correspond alors à la somme des alignements deux à deux, pour toutes les paires de séquences que l'on peut former, fonction communément appelée *somme des paires*. Dans ce cas, la complexité temporelle exprimée en fonction de T, le nombre de séquences de longueurs L que l'on aligne devient $\mathcal{O}(L^T 2^T)$ et $\mathcal{O}(L^T)$ pour la complexité spatiale. Le terme (2^T) correspond au $(2^T - 1)$ provenances possibles qu'il faut évaluer dans la relation récursive (Équation 3.1). Ce type d'approche n'est donc pas réalisable pour plus de trois à quatre séquences. Le programme MSA (Lipman et al., 1989) est une implémentation d'un algorithme heuristique non-exact (Carrillo and Lipman, 1988) basé sur le même principe que le programme BLAST. Cette approche consiste à identifier des zones de l'espace $N-$dimensionnel qui ont peu de chance de contribuer à la solution optimale, et qui peuvent être éliminées du processus de calcul. Cette approche permet d'aligner jusqu'à dix séquences. La qualité de l'heuristique dépend de la similarité entre ces séquences. Ce programme a ensuite servi de base à une nouvelle approche DCA (Stoye et al., 1997) (Divide and Conquer Algorithm) qui divise l'ensemble de séquences en plusieurs ensembles plus petits, pouvant chacun être alignés avec MSA. La difficulté de cette approche consiste à choisir judicieusement les sous-ensembles à former.

3.4.2 L'alignement multiple progressif

Les approches les plus connues concernent les algorithmes dits progressifs (approche gloutonne), qui construisent l'alignement multiple en assemblant des sous-alignements (alignements multiples impliquant un nombre de séquences inférieur à T) par programmation dynamique, un choix d'appariement n'étant jamais remis en cause. Le point commun entre ces méthodes est qu'elles partent d'une collection d'appariements deux à deux de séquences, cette collection étant obtenue par programmation dynamique (alignements globaux et/ou locaux de toutes les séquences prises deux à deux). Dans le cas le plus simple, les $T(T-1)/2$ alignements globaux de deux séquences sont effectués par l'algorithme de N&W. Ensuite, en se basant sur les scores d'alignements obtenus, un arbre évolutionnaire binaire est construit, lequel servira à déterminer l'ordre d'introduction des séquences dans l'alignement multiple. Les séquences sont représentées chacune dans une feuille de l'arbre et chaque noeud interne contient les séquences représentées dans ses deux fils, et toutes les séquences seront finalement représentées par la racine. Comme nous l'avons dit précédemment, cet arbre donne l'ordre dans lequel les alignements deux à deux doivent être effectués. De cet ordre dépend fortement la qualité de l'alignement multiple final si les séquences ne présentent pas une forte similarité. L'alignement multiple est alors élaboré en effectuant un alignement deux à deux dans chacun des noeuds de l'arbre et en partant des feuilles. Chaque noeud interne contient donc l'alignement du contenu de ses deux noeuds fils. Ces deux noeuds peuvent contenir deux séquences seules, ou une

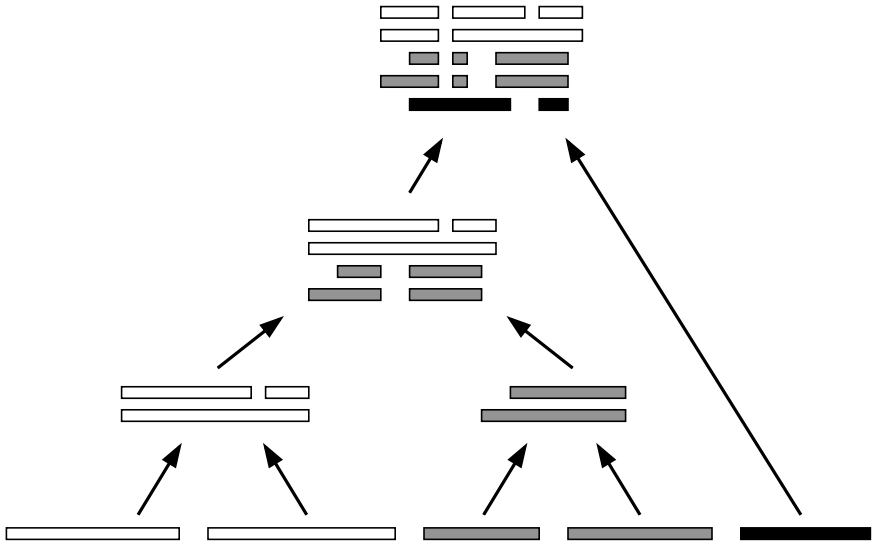


Fig. 3.3 : Cette figure illustre le procédé d'alignement multiple dit progressif. L'alignement multiple est construit de façon gloutonne par une succession d'alignements deux à deux, un choix n'étant jamais remis en cause. La majeure partie des programmes d'alignements multiples utilisent ce procédé. Ils diffèrent par contre pour le système de score utilisé, ainsi que pour la méthode de construction de l'arbre guide.

séquence et un alignement déjà effectué, ou encore deux alignements (Figure 3.3). Les appariements déjà déterminés dans les noeuds inférieurs ne sont jamais remis en cause. Cette procédure cherche généralement à optimiser la somme des paires. Dans ce cas, le système de score s'exprime de la façon suivante : soit a et b deux alignements (multiples ou non), et a_i, b_j donnent respectivement les i^{emes} et j^{emes} positions dans ces alignements. Soit $\mathcal{C}_{i,j}$ l'ensemble de tous les appariements deux à deux que l'on peut former entre les positions a_i et b_j des deux alignements. Le score d'appariement entre a_i et b_j est donné par :

$$\sum_{(x,y)\in\mathcal{C}_{i,j}} B(x,y)$$

Un des premier programme d'alignement multiple par construction progressive à été décrit dans (Feng and Doolittle, 1987).

3.4.3 ClustalW

ClustalW (Thompson et al., 1994; Higgins et al., 1996) est certainement la méthode d'alignement multiple la plus populaire. Elle apporte des améliorations à une première version appelée ClustalV (Higgins et al., 1992). La stratégie de ClustalW consiste à aligner les $n(n-1)/2$ paires de séquences possibles avec l'algorithme de N&W. Les scores de ces alignements sont ensuite convertis en distances évolutionnaires en utilisant la méthode de (Kimura, 1983). On obtient donc une demi-matrice de distance entre toutes les paires de séquences. Une distance évolutionnaire est préférée au score brut des alignements car elle est indépendante de la longueur des séquences. L'arbre binaire guide est construit par une méthode heuristique non exacte qui consiste à agglomérer successivement les séquences les plus proches ("neighbour-joining clustering" (Saitou and Nei, 1987). Nous ne nous attar-

derons pas sur les détails de la construction de l'arbre ainsi que sur le calcul des distances évolutionnaires. Le lecteur intéressé pourra trouver une description détaillée dans (Durbin et al., 1998). Les sous-alignements de l'arbre sont alignés, depuis les feuilles jusqu'à la racine selon le même principe que décrit précédemment.
Le score utilisé correspond à la *somme des paires pondérées*. Ce score cherche à éliminer le problème lié à une sur-représentation d'une sous-famille de séquences, en attribuant à chaque séquence une pondération inversement proportionelle à sa similarité avec les autres séquences.
Le programme inclut plusieurs autres astuces comme l'utilisation de matrices de substitution différentes selon la similarité des séquences qu'il aligne, ou un coût d'indel spécifique à la position dépendant de la composition local en acide aminé.

3.4.4 Alignements multiples utilisant un score basé sur la consistance

D'une façon générale, on peut considérer la stratégie progressive sur le score de la façon suivante. Une collection $\mathcal{C}$ d'appariements deux à deux entre tous les symboles de séquences différentes est obtenue par programmation dynamique. Cette collection est déjà un résultat en soi, mais il est fort probable que des contradictions empêchent de faire figurer tous ces appariements dans un alignement multiple. Si nous supposons par exemple trois séquences A, B et C, alignées deux à deux, on obtient trois alignements possibles. Supposons maintenant trois positions dans ces séquences a_i, b_j et c_k chacune correspondant à un symbole. Si a_i et b_j sont appariés, qu'il en va de même pour b_j et c_k, mais que a_i et c_k ne sont pas appariés, il y a une contradiction qui empêche de représenter ces trois appariements dans un même alignement multiple. Le programme doit donc faire un choix sur les appariements qui lui semblent les meilleurs. Dans le cas de la stratégie progressive sur la fonction de la somme des paires pondérée ou non, ce choix est implicitement basé sur la similarité des séquences qui va déterminer l'ordre de leurs introductions dans la construction de l'alignement multiple.

Une alternative consiste à examiner explicitement ces choix sur la base de la consistance des appariements de $\mathcal{C}$. Cette approche consiste à considérer en priorité les appariements qui sont confirmés par d'autres appariements. Si, pour reprendre notre exemple précédent, la position a_i est appariée avec c_k, les appariements entre a_i, b_j et c_k sont dit consistants trois à trois. Le principe général de ces méthodes consiste à examiner tous les triplets d'appariements et à attribuer plus d'importance aux appariements faisant partie d'un triplet consistant. Ces techniques se limitent à examiner les triplets uniquement, car une généralisation au k-tuplet devient trop lourde pour $k > 3$. En effet, la complexité temporelle de cette opération en fonction de k est donnée par $\mathcal{O}(T^k L^2)$ (Notredame et al., 2000). Il existe plusieurs programmes qui utilisent ce concept, lequel apporte un avantage pour la signification biologique des alignements obtenus. Les deux approches les plus connues sont DiAlign2 (Morgenstern et al., 1993; Morgenstern, 1999) qui produit des alignements multiples basés sur des similarités locales de segments sans indels, et T-COFFEE (Notredame et al., 2000), que nous décrivons ci-dessous en détail.

T-COFFEE

T-COFFEE (Notredame et al., 2000) est un algorithme d'alignement progressif qui cherche à optimiser une fonction appelée COFFEE (Notredame et al., 1998) basée sur la consistance. Ce programme commence par construire une collection, appelée *librairie primaire* d'appariements de positions impliquant deux séquences différentes. Cette librairie est obtenue par deux sources impliquant les $T(T-1)/2$ alignements globaux et locaux deux à deux. Chaque appariement reçoit une pondération qui est fonction de l'identité des deux séquences auquel il appartient. Les deux sources d'appariement sont combinées dans la librairie primaire. Si deux appariements sont identiques, un seul est représenté dans la librairie, mais avec une pondération égale à la somme de celle des deux appariements. Cette libraire peut également inclure d'autres sources d'appariements selon le même principe, comme des appariements issus d'une comparaison structurelle de deux protéines, ou encore issus d'une connaissance experte. On peut de cette manière apporter une contrainte forte sur certains appariements de positions en leurs attribuant un poids élevé.
L'étape suivante, appelée *extension de la librairie*, consiste à examiner les consistances des triplets d'appariements. Si un appariement donné fait partie d'un triplet consistant, sa pondération est augmentée de la pondération minimum des trois appariements du triplet. Finalement, l'alignement multiple est construit par la stratégie progressive décrite précédemment. L'arbre guide est déterminé par la méthode de (Saitou and Nei, 1987), sur une matrice de distance calculée en effectuant $T(T-1)/2$ alignements deux à deux globaux en utilisant comme score les pondérations de la librairie étendue. L'alignement multiple est construit par programmation dynamique en remontant dans l'arbre, jusqu'à la racine. Lorsque deux alignements fixés sont alignés, le score d'appariement utilisé est la pondération moyenne des appariements entre les colonnes considérées. Une différence importante avec le score de la somme des paires "classique" est qu'un appariement entre deux symboles peut recevoir un score différent selon sa position dans l'alignement, car ce score ne dépend plus de la matrice de substitution utilisée, mais de l'ensemble des sources qui l'ont produit, ainsi que de sa consistance. Une extension appelée 3DCOFFEE (O'Sullivan et al., 2004), permet d'inclure dans la librairie primaire une information issue d'alignements structuraux. Cette possibilité relève une différence importante d'avec les autres méthodes, qui ne se contentent que de l'information issue de la séquence primaire.

3.5 Alignements locaux multiples sans indels

Nous allons à présent nous intéresser à une classe particulière des alignements multiples : les alignements locaux multiples sans indels, que nous abrégeons ULMA (pour "Ungapped Local Multiple Alignement"), lesquels sont dans la grande majorité des cas obtenus par un processus d'optimisation par voisinage, bien qu'ils ne soient que très rarement décris sous cet angle. Ce type d'alignement est celui qui nous intéresse particulièrement dans ce travail. Nous décrivons dans cette section le fonctionnement général des quatre approches les plus connues, sur lesquelles sont basées la grande majorité des programmes de découverte d'ULMA. Ces approches seront illustrées par les programmes suivants : le Gibbs Site Sampler (GSS) (Lawrence et al., 1993), qui correspond essentiellement à un grimpeur stochastique, le Gibbs Motif Sampler (GMS) (Neuwald et al., 1995), un algorithme statistique qui effectue un $k-$partitionnement, MEME (Bailey and Elkan, 1995) une procédure d'optimisation des paramètres d'un modèle utilisant une procédure statistique appelée "expectation-maximization", et finalement CONSENSUS (Hertz and Stormo, 1999) qui utilise la stratégie gloutonne de construction de recherche par faisceau.

La Section 4 présentera une description détaillée de la nature des ULMA ainsi que du système de score qui leur est associé. Pour les descriptions qui vont suivre, nous décrirons simplement un ULMA $A = \{a_1, a_2, \ldots, a_N\}$, comme une collection de facteurs a_i de S, dont le nombre N ainsi que la longueur W sont généralement fixées par l'utilisateur. Le j^{eme} symbole d'un facteur a_i est noté $a_{i,j}$. Dans le cas le plus simple, l'ULMA est composé d'exactement un facteur par séquence de S, le nombre de facteurs est donc implicitement fixé par T, le nombre de séquences. Cependant, des ULMAs plus généraux autorisent l'alignement de plusieurs facteurs issus d'une même séquence, ou permettent de ne pas considérer toutes les séquences de S dans l'alignement, certaines pouvant ainsi être identifiées comme ne contenant pas de résidus homologues avec les autres. Cette observation relève une différence majeure par rapport aux alignements multiples par programmation dynamique, lesquels imposent à chaque séquence de contribuer exactement une fois à l'alignement. Toutes les méthodes de découverte d'ULMAs ont un système de score basé sur l'entropie relative, ou le ratio du logarithme des vraisemblances, deux systèmes qui correspondent en fait à deux points de vue différents sur une même mesure. Le premier considère ce score en fonction de la théorie de l'information (Shannon, 1948) et l'utilise pour mesurer le degré de conservation dans les colonnes de l'ULMA. Le deuxième point de vue est statistique et considère ce score comme un ratio de probabilités : la probabilité d'observer l'alignement selon son modèle, divisé par la probabilité de l'observer selon un modèle nul. Nous rendons le lecteur attentif au fait que l'estimation de ces probabilités peut être effectuée de différentes manières par chaque programme de découverte d'ULMAs. Nous ne décrivons que le principe général. Toutes ces méthodes associent à l'ULMA deux modèles, F pour celui qui représente l'ULMA et F^0 pour le modèle nul, généralement appelé *modèle du bruit de fond*. F consiste en une matrice position spécifique qui décrit les probabilités d'observer les symboles dans chacune des colonnes de l'ULMA. Les valeurs de F sont notées $f_{i,j}$ avec $i \in \{1..W\}$ représentant les colonnes de l'ULMA et $j \in \{1..K\}$ représentant les différents symboles, ou résidus. $f_{i,j}$ correspond à la probabilité d'observer le symbole j dans la i^{eme} colonne de l'ULMA. Cette probabilité correspondant en fait à une fréquence car elle est estimée à partir du décompte des symboles dans les colonnes de l'ULMA. Une telle matrice est généralement appelée une *matrice de fréquence position spécifique* ou PSFM pour "Position Specific Frequency Matrix". Certaines méthodes ajoutent des *pseudocomptes*, lesquels assurent qu'aucun symbole n'ait une probabilité nulle dans F. Une telle matrice permet de calculer la probabilité qu'un facteur de taille W ait été généré par le modèle. Soit $x = \{x_1, x_2, \ldots, x_W\}$ un facteur de S avec x_i son i^{eme} symbole. La probabilité $P(x|F)$ de générer x par le modèle F est donnée par :

$$P(x|F) = \prod_{i=1}^{W} f_{i,x_i}$$

Le modèle du bruit de fond F^0 donne pour chaque symbole i la probabilité f_i^0 de l'observer par hasard. Ces probabilités f_i^0 sont estimées à partir des fréquences de chaque symbole i dans S, en ajoutant éventuellement des pseudocomptes. Il est courant de proposer à l'utilisateur de spécifier lui-même ces probabilités en fonction de connaissances qu'il peut avoir sur ses données. De la même manière que pour F, la probabilité $P(x|F^0)$ d'un facteur x selon le modèle F^0 est calculée par :

$$P(x|F^0) = \prod_{i=1}^{W} f_{x_i}^0$$

A) PSFM : Matrice de fréquences position-spécifiques

A	0.06	0.88	0.00	0.90	0.56	0.97	0.27	0.68
T	0.91	0.11	0.97	0.08	0.44	0.03	0.66	0.14
C	0.02	0.01	0.02	0.00	0.00	0.00	0.01	0.08
G	0.01	0.01	0.01	0.02	0.00	0.00	0.06	0.09

B) PSSM : Matrice de poids (log-odds) position-spécifiques

A	-2.36	1.49	....	1.51	0.82	1.62	-0.22	1.12
T	1.65	-1.44	1.74	-1.83	0.61	-3.24	1.19	-1.02
C	-3.59	-5.59	-3.59	....	....	....	-5.59	-1.59
G	-4.88	-4.88	-3.88	-2.88	....	....	-1.29	-0.71

Fig. 3.4 : Matrices position spécifiques obtenues par décompte des symboles des boîtes TATA issues de 196 promoteurs de plantes. (A) La première matrice contient simplement les fréquences observées des nucléotides (A,T,C,G) pour les huit positions de la boîte TATA. (B) Cette matrice tient en plus compte du "bruit de fond" correspondant à la fréquence attendue des nucléotides. Certaines valeurs de cette matrice sont indéterminées si $f_{i,j} = 0.0$. Dans la pratique, afin d'éviter ces indéterminations, des pseudocomptes sont ajoutés aux décomptes de chaque symbole pour s'assurer que toutes les valeurs de F soient toujours strictement supérieures à 0.0.

La distribution F^0 permet de tenir compte du biais compositionnel de S, et on attend des facteurs qui composent l'ULMA une forte probabilité qu'ils aient été générés par F, et une faible probabilité qu'il ait été générés par F^0. L'ULMA est alors modélisé par une matrice LLR, (pour "Log Likelihood Ratio"), ou chaque valeur $llr_{i,j} = \log(f_{i,j}/f^0_j)$. Ces matrices sont souvent appelées des PSSM pour "Position Specific Scoring Matrix". Le logarithme permet d'obtenir un score additif. Le score $LLR(x)$ d'un facteur x est alors donné par :

$$LLR(x) = \sum_{j=1}^{W} llr_{j,x_j}$$

Le score de l'ULMA A sur lequel est estimé F correspond à la somme des scores de chacun de ses facteurs, soit :

$$LLR(A) = \sum_{i=1}^{N} \sum_{j=1}^{W} llr_{j,a_{i,j}} \tag{3.2}$$

Ce score correspond à un ratio du logarithme des vraisemblance, et comme nous le verrons dans la Section 4, mesure également un degré de conservation des symboles dans les colonnes de l'ULMA lorsqu'il est considéré du point de vue de la théorie de l'information. La figure 3.4 propose un exemple de ces modèles PSFM et PSWM calculés par simple décompte de symboles dans un alignement de la boîte TATA de 196 promoteurs de plantes issus de la bases de donnée EPD (Eucariotic Promotor Database) (Schmid et al., 2004).

Il existe également des programmes basés sur d'autres types de modèles mais nous ne les traiterons pas en détail. Par exemple, un modèle décrit dans (Keich and Pevzner, 2002) étend la PSFM en représentant toutes les corrélations possibles entre les colonnes de la

matrice prises deux à deux. Ce modèle plus informatif implique de calculer les probabilités de tous les dinucléotides (seize en tout) pour les $\binom{l}{2}$ paires de positions dans le modèle. Une autre approche décrite dans (Zhou and Liu, 2004) est basée sur le même principe, mais ne considère que les paires de corrélations jugées significatives, permettant ainsi d'alléger considérablement le modèle et d'éviter "l'overfitting". Ces deux méthodes ne sont utilisées que sur des séquences nucléiques ou $K = 4$. Il apparaît effectivement difficile des les étendre sur un alphabet de taille plus élevée comme celui des protéines.

3.5.1 Le Gibbs Site Sampler

Le Gibbs Site Sampler (GSS) (Lawrence et al., 1993) est un programme d'optimisation d'ULMA dont l'implémentation la plus récente est conjointe avec celle du Gibbs Motif Sampler (GMS) (Neuwald et al., 1995), résultant souvent en une confusion entre ces deux méthodes. Le GSS alterne deux phases appelées *phase prédicative* et *phase de mise à jour*. Ce procédé est très similaire à celui utilisé par l'algorithme EM. Son principe consiste à faire évoluer alternativement un alignement et les modèles F et F^0. Durant la phase prédicative, l'alignement est remis en cause sur le modèle et est éventuellement modifié. Les modèles F et F^0 sont ensuite mis à jour sur le nouvel alignement. Au début, un ULMA A est initialisé en choisissant aléatoirement un facteur a_i par séquence s_i. L'ULMA est ainsi constitué de T facteurs de S. Les modèles F et F^0 sont calculés à partir des décomptes de symboles et de pseudocomptes. Une particularité réside dans le fait que F^0 n'est estimé qu'à partir du décompte des symboles de S qui sont exclus l'ULMA. F^0 est donc modifié au cours des itérations.

- *La phase prédicative.* Cette phase consiste à remettre en cause un facteur choisi a_i, lequel est retiré de l'ULMA. Les modèles F et F^0 sont alors modifiés en conséquence, en faisant passer les décomptes des symboles de a_i depuis F vers F^0. L'appartenance potentielle de chaque facteurs x de s_i est prédite sur la base d'un score correspondant à $P(x|F)/P(x|F^0)$ (le logarithme n'est pas utilisé contrairement au score $LLR(x)$, pour une raison présentée plus loin. Ce score est associée à chaque facteur de s_i. Une nouvelle occurrence a_i est déterminée parmi l'ensemble des facteurs x en utilisant un tirage aléatoire biaisé sur les scores obtenus. Ce tirage correspond à la sélection de type "roue de loterie" utilisée par les algorithmes génétiques. Le facteur ainsi choisi est inséré dans l'ULMA.

- *La phase de mise à jour.* Les modèles F et F^0 sont mis à jour en fonction du nouvel ULMA.

Ce processus est répété pour tous les facteurs a_i de l'ULMA, en les choisissant aléatoirement sans remise, ou dans un ordre prédéterminé. Ensuite, une étape appelée *phase shift* est effectuée pour corriger un éventuel "mauvais calage" de l'ULMA. Il arrive en effet fréquemment que les facteurs qui le composent soient situés quelques positions à coté d'une zone plus significative dans S. Le programme cherche alors à corriger directement un tel mauvais calage de la façon suivante : Tous les ULMA A' obtenus pas un décalage simultané de la position des facteurs, dans les limites de quelques positions vers la droite ou la gauche sont évalués par $LLR(A')$, puis un tirage aléatoire biaisé en fonction du score choisit un nouvel ULMA parmi les décalages évalués. Toutes les opérations décrites jusqu'à maintenant constituent un *cycle* de l'algorithme. A la fin de chaque cycle, l'ULMA A est évalué

par $LLR(A)$ F^0 étant toujours estimé à partir du décompte des symboles extérieurs à l'ULMA, et en ajoutant des pseudocomptes. Les cycles sont répétés jusqu'à ce qu'aucune amélioration ne soit observée pendant un nombre de cycles fixé. Le programme répète tout le processus sur un nombre de graines donné et produit en sortie le meilleur ULMA trouvé accompagné de ses modèles F et F^0. D'autres options sont proposées comme par exemple de pouvoir spécifier le nombre de facteurs issus de chaque séquence s_i (nombre qui doit être donné séparément pour chaque séquence). Ce programme propose également un moyen permettant de faire varier la largeur[2]. Pour cela, il échantillonne aléatoirement W colonnes dans un ULMA plus large, et ne considère que celles-ci pour l'évaluation. Les W colonnes considérées sont choisies par un tirage biaisé et sont continuellement remises en cause. Cette opération est décrite dans (Neuwald et al., 1995) sous le nom de "fragmentation model".

Ce mode d'optimisation originalement décrit sous un point de vue statistique correspond à un grimpeur stochastique sur l'espace des ULMA et utilisant deux fonctions de voisinage. Le premier voisinage correspond à tous les ULMAs qui ne diffèrent que d'un facteur et le deuxième voisinage correspond à quelques ULMA décalés. Une particularité est que le voisinage n'est pas directement évalué sur OF mais sur le ratio de probabilité $P(x|F)/P(x|F^0)$, où F et F^0 sont estimés sur un ULMA auquel un facteur a été retiré. Cette observation signifie que le choix du voisin est effectué en fonction d'un score qui n'est pas strictement équivalent à la fonction objectif $LLR(A)$. La différence est cependant minime et il est vraisemblable qu'elle soit sans conséquence dans la grande majorité des cas. Comme nous l'avons dit précédemment, le choix du voisin est fonction d'un score qui n'utilise pas le logarithme comme pour LLR. La raison n'est pas donnée dans les publications, mais nous avons pu observer que l'utilisation du logarithme réduit les écarts de valeurs entre les différents facteurs évalués, et la fonction de sélection de type "roue de loterie" n'exerce alors pas une pression sélective suffisante pour permettre à l'algorithme de converger.

Un programme appelé BIOPROSPECTOR (Liu et al., 2001) propose une extension du principe, laquelle est spécifique aux sites de régulation dans les séquences nucléiques. Ce programme permet d'optimiser simultanément deux ULMAs séparés par une région variable. Une autre utilisation est décrite dans (Nielsen et al., 2004), qui applique le principe du GSS pour aligner des régions antigèniques de plusieurs centaines de protéines courtes. La méthode décrite apporte plusieurs modifications, mais la plus importante concerne certainement un système de pondération des positions dans les séquences, permettant d'apporter un *a priori* sur la position approximative des facteurs que l'on cherche à aligner.

3.5.2 MEME

MEME (Bailey, 1995; Bailey and Elkan, 1995) est un programme d'optimisation basé sur l'algorithme statistique "expection-maximization" (EM)(Gelman et al., 2004). MEME est basé sur un premier programme décrit dans (Lawrence and Reilly, 1990), lequel permet de produire des ULMAs uniquement formé d'exactement un facteur par séquence. MEME reprend le même principe d'optimisation mais propose différents modes : le mode OOPS

[2]Le paramètre W spécifiant la longueur des facteurs est vu comme une *largeur* lorsque l'on considère l'ULMA dans son ensemble.

(One Occurrence Per Sequence), identique à la première version décrite. Le mode ZOOPS (Zero or One Occurrence Per Sequence) qui comme son nom l'indique permet à un nombre donné de séquences de ne pas contribuer à l'ULMA, et finalement le mode TCM (Two-Component Mixture) qui permet à chaque séquence de contribuer zéro, une ou plusieurs fois à l'alignement. Lorsqu'une séquence contribue plus d'une fois, les facteurs ne sont pas autorisés à se recouvrir. Les paramètres W et N doivent être donnés par l'utilisateur. Le paramètre W peut être donné soit exactement, ou alors sous la forme d'un intervalle de valeurs. Pour cette dernière possibilité, le programme utilise une estimation statistique pour déterminer la valeur la plus pertinente. Le principe de l'application de l'algorithme EM présente des similarités avec GSS, par le fait que deux phases sont itérativement alternées. Une phase où un ULMA est mis en cause sur un modèle, et une phase où le modèle est mis à jour. Par contre, les calculs mis en oeuvre diffèrent beaucoup. La première phase (*expectation*) consiste à calculer pour tous les facteurs de S une probabilité qu'ils fassent partie de l'ULMA. Ces probabilités sont calculées à l'aide des modèles F et F^0 en utilisant le théorème de Bayes (nous ne nous attarderons pas ici sur ces calculs). La deuxième phase appelée *maximization*, met à jour les paramètres du modèle, faisant en sorte qu'ils maximisent la vraisemblance des probabilités calculées à l'étape précédente. Contrairement au GSS, MEME ne maintient pas un ULMA donné, mais maintient pour tous les facteurs de S la probabilité qu'ils fassent partie de l'ULMA. L'alignement produit en sortie sera composé des facteurs qui maximisent ces probabilités. Les deux phases consistent en :

- *Expectation :* Pour chaque facteur de longueur W de S , MEME calcule la probabilité qu'il fasse partie de L'ULMA sachant les modèles F et F^0. Si c'est la première itération, les modèles sont initialisés, soit par l'utilisateur, soit par une procédure qui nous décrirons par la suite.

- *maximization :* Cette étape met à jour les paramètres des modèles. Les décomptes de symboles de tous les facteurs de S sont ajoutés au modèle, mais en les pondérant directement par la probabilité calculée à l'étape précédente. Les facteurs ayant obtenu une probabilité faible (souvent très proches de zéro) auront de ce fait un poids négligeable dans les nouveaux modèles.

Ces deux étapes sont itérées jusqu'à une condition d'arrêt qui est fonction de la stabilité du modèle obtenu. L'ULMA est produit en identifiant pour chaque séquence, le facteur qui maximise la probabilité sachant les modèles.

L'étape d'initialisation des modèles est implémentée de façon déterministe. Le programme commence par produire un modèle F pour chaque facteur de longueur W dans S. Ce modèle attribue une probabilité proche de 1.0 pour chaque symbole du facteur, et ajoute des pseudocomptes pour les autres symboles, de façon à ce que les valeurs somment à 1.0. Chaque modèle est optimisé par trois itérations de l'algorithme EM. Celui qui converge le plus est choisi comme graine et son optimisation est poursuivie jusqu'à ce que ses paramètres se stabilisent. Ce modèle et l'alignement correspondant sont donnés en résultat. Il est montré que cette procédure cherche à optimiser le logarithme du ratio des vraisemblances $LLR(A)$ (Équation 3.2). Le principe du programme même peut être vu comme un grimpeur strict.

3.5.3 Le Gibbs Motif Sampler et dérivés

Le Gibbs Motif Sampler (GMS) (Neuwald et al., 1995) est une méthode permettant de découvrir un nombre donné d'ULMAs, chacun composé d'un nombre de facteurs estimés par le programme. Cette estimation est basée sur une valeur attendue, spécifiée par l'utilisateur pour chacun des ULMAs. Le principe utilisé est repris dans plusieurs programmes dédiés aux séquences nucléiques pour la recherche de sites de fixation de facteurs de transcription. Le Gibbs Recursive Sampler (Thompson et al., 2003) est dédié à la découvertes de sites multiples dans des séquences nucléiques de composition très hétérogènes. AlignAce (Hughes et al., 2000) est spécifique aux génomes de levures. La différence avec la méthode originale consiste essentiellement en deux points : un modèle de bruit de fond spécifique pour les levures et une stratégie consistant à optimiser plusieurs ULMAs successivement plutôt que simultanément. Une méthode décrite dans (Thijs et al., 2001) utilise un modèle de bruit de fond basé sur une chaîne de Markov d'ordre supérieur à un, qui permet de modéliser la probabilité d'un symbole en fonction de ceux qui le précèdent. Cette approche permet de baisser le niveau de bruit pour la détection de site de fixation de facteurs de transcription.

Soit z, le nombre d'ULMAs désirés, notés $A_1, A_2, \ldots, A_z$. Le principe général consiste à effectuer un $z-$partitionnement des facteurs de S. Ainsi, A_i peut être vu comme une partie et nous utiliserons la notation A^0 pour spécifier la partie des facteurs qui correspondent au bruit de fond, bien qu'ils ne fassent pas partie d'un ULMA. Pour simplifier, nous supposons que l'utilisateur spécifie la même valeur W pour tous les ULMAs[3]. Soient $N_1, N_2, \ldots, N_z$ le nombre de facteurs attendus par l'utilisateur pour chacun des ULMAs. Le nombre effectif est cependant ajusté en cours d'optimisation par le biais de e_i, une probabilité *a posteriori* qu'un facteur quelconque fasse partie de A_i. Cette probabilité est estimée en fonction du nombre courant de facteurs dans A_i ainsi que de N_i. Le programme crée alors $z+1$ modèles : $F^0, F^1, \ldots, F^Z$, avec F_i pour le modèle de l'ULMA A_i et F^0 pour le modèle du bruit de fond.

Le programme est initialisé en créant z ULMAs composés d'un nombre N_i de facteurs aléatoires non recouvrant et de taille W. Deux étapes sont itérativement effectuées : (1) Un facteur de taille W est choisi dans S (lors des itérations successives, les facteurs sont choisis dans leur ordre d'apparition dans S). Si le facteur choisi fait déjà partie d'un ULMA A_i, il en est retiré et le modèle F_i est mis à jour en conséquence. (2) Le facteur choisi est évalué contre chaque partition, par un score composé du ratio des vraisemblances, ainsi que d'une pondération fonction de e_i. Un tirage aléatoire biaisé par ces scores est effectué pour déterminer la partition dans laquelle le facteur doit être inséré, (en incluant A_0, dans le cas où le facteur ne fait partie d'aucun ULMA). La façon de traiter les recouvrements des facteurs n'est malheureusement pas décrite dans la publication.

Dans la pratique, le nombre de facteurs dans les ULMAs baisse rapidement. Il faut alors qu'un minimum de deux facteurs similaires soient placés dans le même ULMA A_i pour que son modèle F_i devienne significatif. Les autres facteurs similaires seront ensuite rapidement identifiés comme appartenant à la partition A_i.

[3]Le paramètre W doit également être donné pour chacun des ULMA, mais la largeur effective peut être ajustée par le programme selon le "framentation model" décrit dans la Section 3.5.1.

3.5.4 Consensus

CONSENSUS (Hertz et al., 1990; Hertz and Stormo, 1999) est un algorithme de recherche par faisceau pour la découverte d'un ULMA de largeur W et d'un nombre d'occurrences N fixé par l'utilisateur. Son fonctionnement par défaut permet de trouver un ULMA composé d'exactement un facteur par séquence, mais cette contrainte peut être relâchée de façon à ce que chaque séquence contribue au moins une fois à l'ULMA, ou encore de façon à ce que chaque séquence contribue zéro, une ou plusieurs fois à l'ULMA. La fonction objectif optimisée correspond exactement à LLR, avec les modèles F et F^0 estimés simplement à partir des décomptes des symboles, sans l'ajout de pseudocomptes. F^0 est déterminé au début de la construction et reste inchangé durant tout le processus. Cette fonction correspond à l'*entropie relative*, notée $OF^{(H)}$, laquelle est décrite en détail dans la Section 4.4. La publication de 1999 décrit également une statistique sur le score d'entropie permettant d'attribuer à un ULMA et pour des paramètres W et N donnés, une p-valeur, qui correspond à la probabilité d'observer un score supérieur ou égal sur un alignement aléatoire. Cette p-valeur peut servir à comparer des ULMA obtenu sur un même ensemble S mais avec différents paramètres W et N afin d'identifier le plus significatif du point de vue de la statistique utilisée.

Nous décrivons le principe de l'algorithme pour son fonctionnement par défaut, forçant chaque séquence à contribuer exactement une fois à l'ULMA. L'option permettant de relâcher cette contrainte fonctionne sur le même principe. Soient $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_T$ avec $\mathcal{F}_i$, l'ensemble des facteurs de s_i, et soit Q le paramètre spécifiant la largeur du faisceau (fixé par défaut à $Q = 1000$). Soit $\mathcal{A}_2, \ldots, \mathcal{A}_T$ des listes d'ULMAs, où les ULMAs d'une liste $\mathcal{A}_i$ sont formés de i facteurs. La taille de ces listes est limitée au nombre Q. La méthode consiste à générer successivement les listes $\mathcal{A}_2, \ldots, \mathcal{A}_T$ en $T-1$ étapes. Le programme commence par former successivement les $|\mathcal{F}_1| \cdot |\mathcal{F}_2|$ ULMAs possibles que l'on peut construire en choisissant un facteur respectivement dans $\mathcal{F}_1$ et $\mathcal{F}_2$. Ces ULMAs sont évalués au fur et à mesure avec $OF^{(H)}$, et les q meilleurs sont mémorisés dans $\mathcal{A}_2$ jusqu'à concurrence du nombre Q. Au début de la construction on a généralement $|\mathcal{A}| < Q$. Ensuite, pour chaque nouvelle étape $t \in \{3..T\}$ de la construction, les $|\mathcal{F}_t| \cdot |\mathcal{A}_{t-1}|$ ULMAs pouvant être formés par l'ajout d'un facteur de $\mathcal{F}_t$ dans un ULMA de $\mathcal{A}_{t-1}$ sont successivement évalués et comme précédemment, les meilleurs sont mémorisés dans $\mathcal{A}_t$. La liste $\mathcal{A}_{t-1}$ peut alors être supprimée. Finalement, le meilleur ULMA de $\mathcal{A}_T$ est produit en sortie. Le programme peut être paramétré pour produire également les meilleurs ULMA des listes $\mathcal{A}_i$ pour des i donnés. Le paramètre Q est essentiel pour la qualité de la recherche, mais sa valeur est vite limitée par la mémoire de la machine. CONSENSUS fonctionne raisonnablement bien quand l'ULMA à découvrir est composé de facteurs très similaires. Ce programme souffre par contre d'une limitation importante due à l'approche gloutonne qu'il utilise. Nous avons pu observer que ses performances chutent rapidement dès que les séquences sont moins conservées.

Chapitre 4

Une approche pour l'alignement local multiple : définitions, fonctions objectif et optimisation

4.1 Introduction

Ce chapitre est consacré à la définition de l'alignement local multiple et sans indel (que nous abrégeons ULMA pour *Ungapped Local Multiple Alignment*), ainsi que deux fonctions objectif permettant de leur attribuer une mesure de qualité. La première fonction est l'*entropie relative* (aussi connue sous le nom *divergence de Kullback-Leibler*), laquelle a été largement utilisée par la communauté. Nous décrirons également une nouvelle fonction appelée *entropie relative recouvrante*. Cette nouvelle fonction, contrairement à l'entropie relative qui considère tous les symboles comme étant distincts, est capable de prendre en compte une mesure de "recouvrement" entre les différents symboles, ce qui la rend particulièrement bien adaptée pour mesurer la qualité d'un alignement de protéines. Cette fonction augmente sensiblement la pertinence de la mesure ainsi que les performances d'optimisation.
Le chapitre est subdivisé de la façon suivante : la Section 4.2 définit la nature de l'ULMA ainsi que les notations qui y sont associées. La Section 4.3 donne un aperçu de la notion d'entropie au sens de la théorie de l'information. La Section 4.4 traite de l'entropie relative comme fonction objectif, fonction qui est largement utilisée dans le domaine des ULMA. La Section 4.5 traite de notre nouvelle fonction : l'entropie relative recouvrante. Finalement, la section 4.6 décrit quatre modèles de contraintes sur la répartition des occurrences de l'ULMA dans l'ensemble de séquences. Un grimpeur strict permettant l'optimisation sur chacun des modèles de contraintes est décrit. Nous présentons également deux stratégies permettant de générer des graines prometteuses pour le modèle de contraintes le plus simple.

4.2 L'alignement local multiple sans indels

4.2.1 Introduction

Un alignement local multiple sans indels (ULMA) est essentiellement une collection de N facteurs de taille fixée W, non recouvrants, lesquels sont issus d'un ensemble de

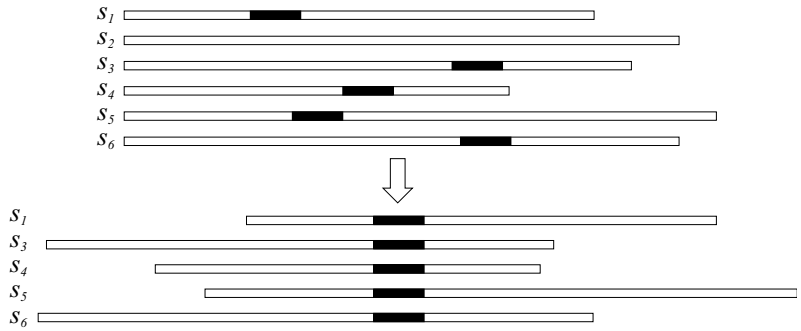


Fig. 4.1 : Ce schéma représente un alignement de cinq *occurrences* issues d'un ensemble de six séquences.

séquences supposées homologues ou apparentées. La difficulté consiste à choisir ces N facteurs de façon à ce qu'ils soient le plus conservés (similaires) possibles, le but étant d'identifier des résidus (symboles) homologues dans les séquences, c'est-à-dire qui ont évolués à partir d'un ancêtre commun (lequel est bien entendu inconnu). Les facteurs découverts sont généralement représentés alignés, ce qui permet de disposer les acides aminés (ou les nucléotides) supposés homologues dans une même colonne (Fig 4.1).
Afin de différencier les facteurs constituant l'ULMA de tous les autres présents dans l'ensemble de séquences, nous les appellerons des *occurrences* de l'ULMA. Cette terminologie se justifie par le fait qu'un ULMA est un modèle d'une propriété biologique. Dans ce cas, les portions des séquences impliquées dans cette propriété sont effectivement des occurrences de l'ULMA.
Le nombre d'occurrences N à identifier ainsi que leur longueurs W sont choisies par l'utilisateur et doivent rester inchangées durant le processus d'optimisation. Ces contraintes sont requises par la nature des fonctions objectif que nous utilisons, pour lesquelles seuls des ULMAs composés du même nombre N d'occurrences de mêmes longueur W sont comparables. Lever partiellement ces contraintes peut se faire à l'aide d'une statistique sur le score, permettant d'évaluer la signifiance d'un ULMA en fonction de W et de N.
Un autre aspect important est celui des contraintes d'optimisation. Ces contraintes empêchent qu'un ULMA contienne des occurrences recouvrantes. D'autres contraintes d'optimisation concernant les répartitions autorisées des occurrences dans les séquences peuvent également être utilisées. Le choix d'une contrainte de répartition des occurrences se fera en fonction des connaissances préalables qu'un utilisateur peut avoir sur ses données.

4.2.2 Notations

Soit $S = \{s_1, s_2, \ldots, s_T\}$ un ensemble de T séquences de longueur L avec $s_{i,j}$ étant le j^{eme} symbole de la i^{eme} séquence. Pour simplifier, nous supposons que toutes les séquences ont la même longueur L. S est définit sur un alphabet de taille finie $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_K\}$ ($K = 4$ pour l'ADN et $K = 20$ pour les protéines). Soient N et W des paramètres utilisateur où N est le nombre de facteurs qui composent l'ULMA et W est leur longueur (on parle également de W comme étant la largeur de l'ULMA).
Soit $S^* = \{s_1^*, s_2^*, \ldots, s_{T(L-W+1)}^*\}$ l'ensemble des facteurs de longueur W présents dans S, tel que le facteur s_i^* commence au symbole $s_{\eta(i),\kappa(i)}$, avec $\eta(i) = \lfloor (i-1)/(L-W+1) \rfloor + 1$

et $\kappa(i) = \mathrm{mod}(i-1, L-W+1)+1$ (mod() étant la fonction modulo). $\eta(i)$ correspond donc à l'indice de la séquence contenant le facteur s_i^* et $\kappa(i)$ correspond à la position de ce facteur dans la séquence. Ainsi, si les séquences s_j sont parcourues dans l'ordre donné par j, s_i^* est donc le i^{eme} facteur rencontré dans S. La représentation S^* permet d'éviter de décrire un pseudo facteur à cheval sur deux séquences.
Un ULMA est codé par un vecteur d'entier $V = (v_1, v_2, \ldots, v_N)$ avec $v_i \in \{1, 2, \ldots, T(L-W+1)\}$ correspondant au facteur $s^*_{v_i} \in S^*$. Afin de s'affranchir de la relation avec S et simplifier les notations, nous décrirons les N occurrences d'un ULMA par un ensemble de chaînes de caractères $\{A_1, A_2, \ldots, A_N\}$, avec $A_{i,j}$ étant le j^{eme} symbole de A_i, chaque A_i correspondant au facteur $s^*_{v_i} \in S^*$.

4.3 Notions d'entropie et de contenu en information

Avant de décrire les fonctions objectif basées sur l'entropie ou le contenu en information, nous allons commencer par illustrer succinctement ces notions qui sont issues de la théorie de l'information (Shannon, 1948). Cette théorie a été initialement développée sur des problèmes de communication (transfert d'information) entre une source et un récepteur. Considérons la source comme un générateur de symboles sur un alphabet de cardinal quatre (des nucléotides par exemple). Ce générateur est représenté par une variable aléatoire Z pouvant prendre quatre valeurs $Z \in \{\mathrm{A}, \mathrm{T}, \mathrm{C}, \mathrm{G}\}$ sur une distribution de probabilités Θ avec $\theta_i = P(Z = i)$. L'information apportée par un symbole i émis est définie par $I_\Theta(i) = -\log_2(\theta_i)$, l'unité étant le bit. Plus un symbole est fréquent et moins il apporte de l'information. En effet, si la distribution de probabilité Θ est concentrée en un seul point, (une probabilité de 1.0 pour un symbole i donné et 0.0 pour tous les autres), l'issue d'une émission sera connue à l'avance et ne nous apporte aucune information : $I_\Theta(i) = -\log_2(1.0) = 0$ bit. Réciproquement, si Θ est uniforme, chaque symbole a la même probabilité d'être émis, (soit 1/4). Dans ce cas l'émission d'un symbole i apporte une information de $I_\Theta(i) = -\log_2(1/4) = 2$ bits.
L'espérance $\mathbb{E}$ de l'information obtenue à chaque émission s'appelle l'*entropie* H de la distribution Θ. Elle est définie par :

$$H(\Theta) = \mathbb{E}\left(I_\Theta(Z)\right) = \sum_{i \in Z} \theta_i \log_2\left(\theta_i\right)$$

(Étant donné que $\lim_{\theta \to 0^+} \theta \log(\theta) = 0$, on convient que $\theta \log(\theta) \equiv 0$ si $\theta = 0$).
L'unité utilisée (le bit) peut s'interpréter par le nombre minimum de questions dichotomiques qu'il faut poser en moyenne pour identifier un symbole émis. Ainsi, 1 bit correspond à la quantité d'information (au sens de Shannon) que l'on donne lorsque l'on répond par oui ou par non à une question.
L'entropie est maximisée à $\log_2(|Z|)$ lorsque Θ est uniforme et elle est minimisée à 0.0 lorsque Θ vaut 1.0 pour un θ_i et 0.0 pour tous les autres. L'entropie mesure donc également la dispersion des θ_i.
Supposons maintenant que le prochain symbole émis ne le soie plus selon la distribution Θ, mais selon une autre distribution Φ. On peut dans ce cas définir une mesure d'*information relative* qui s'exprime par la différence :

$$I_{\Phi|\Theta}(i) = I_\Theta(i) - I_\Phi(i)$$

Cette mesure nous donne l'information sous Θ apportée par un symbole émis sous la distribution Φ. Pour illustrer ce propos par un cas extrême, supposons que la loi Φ vaille

1.0 pour un symbole donné (et 0 pour tous les autres) Ceci équivaut à déterminer le symbole i qui va sortir. Donc, son information relative est équivalente à son information sous Θ :

$$I_{\Phi|\Theta}(i) = I_{\Theta}(i) - 0$$

De la même manière que pour l'entropie, nous pouvons à présent définir *l'entropie relative* (également appelée *divergence de Kullback-Leibler* (Kullback, 1968)) comme étant l'espérance de l'information relative :

$$H(\Phi|\Theta) = \mathbb{E}\left(I_{\Phi|\Theta}(Z)\right) = \sum_{i \in Z} \phi_i \log_2 \left(\frac{\phi_i}{\theta_i}\right) \tag{4.1}$$

L'interprétation de cette formule peut se faire de la façon suivante : si des symboles i sont émis selon une distribution Φ, l'information moyenne vaut $H(\Phi)$. Si l'on considère par contre cette espérance *par rapport* au contenu en information selon Θ, on a $\mathbb{E}_{\Phi} I_{\Theta}$. L'entropie relative mesure l'information apportée en plus par cette espérance, soit $\mathbb{E}_{\Phi} I_{\Theta} - H(\Phi)$. Il faut noter ici que l'entropie relative n'est définie que pour des valeurs de θ_i strictement positives.

L'entropie relative mesure une divergence entre deux distributions. On parle de divergence plutôt que de distance car $H(\Phi|\Theta) \neq H(\Theta|\Phi)$. Dans le même contexte, l'entropie $H(\Theta)$ peut se voir comme une mesure de convergence de Θ vers une distribution uniforme. Dans le cadre de notre problème, l'entropie relative est utilisée pour mesurer la divergence entre la distribution des symboles qu'on observe dans une colonne d'un alignement, avec celle que l'on s'attendrait à observer par hasard. La Section suivante décrit comment cette mesure est utilisée sur des alignements multiples.

4.4 L'entropie relative comme fonction objectif

L'*entropie relative* ou divergence de Kullback-Leibler (Kullback, 1968) est la fonction objectif "classique" pour l'optimisation des ULMAs. Elle est utilisée pour sa capacité à mesurer une divergence entre la distribution des symboles que l'on observe dans les colonnes de l'alignement, et la distribution que l'on attendrait par hasard (appelée généralement *distribution du bruit de fond* ou *distribution a priori*). Utiliser une mesure d'entropie simple pourrait également donner de bon résultats si la distribution du bruit de fond est uniforme ou proche de l'uniformité. En effet, maximiser l'entropie relative à une distribution uniforme est strictement équivalent à minimiser l'entropie simple. Cependant, tenir compte du bruit de fond est extrêmement important dans le cas de séquences de biopolymères car cette distribution est alors souvent loin d'être uniforme.

L'entropie relative est fonction des décomptes de symboles dans chaque colonne de l'ULMA concerné, et dépend également d'un vecteur de probabilité bruit de fond. Soit $F^0 = f_1^0, f_2^0, \ldots, f_K^0$ le vecteur des probabilités du bruit de fond de chaque symbole. Ce vecteur de probabilités est assigné directement à partir des fréquences relatives de chaque symbole dans l'ensemble de séquences S. Toutefois, toute autre information que l'on pourrait avoir sur ces probabilités peut également être utilisée (comme des fréquences relatives de symboles dans une base de donnée spécifique à une espèce par exemple). Ce vecteur est déterminé au début de l'optimisation puis reste inchangé durant tout le processus.
Les décomptes des symboles dans les colonnes de l'ULMA sont représentés dans une matrice C :

$A_1 =$ ‘‘YVCCNTDRCN’’
$A_2 =$ ‘‘YVCCSTEKCN’’
$A_3 =$ ‘‘LLCCTTDNCN’’
$A_4 =$ ‘‘VKCCKTDRCN’’
$A_5 =$ ‘‘VMCCKTDKCN’’
$A_6 =$ ‘‘PVCCRTDLCN’’
$A_7 =$ ‘‘VICCSTDKCN’’

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
A	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.29	0.00	0.00
N	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.14	0.00	1.00
D	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.00	0.00	0.00
C	0.00	0.00	1.00	1.00	0.00	0.00	0.00	0.00	1.00	0.00
Q	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
E	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00
G	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
H	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
L	0.14	0.14	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00
K	0.00	0.14	0.00	0.00	0.29	0.00	0.00	0.43	0.00	0.00
M	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
S	0.00	0.00	0.00	0.00	0.29	0.00	0.00	0.00	0.00	0.00
T	0.00	0.00	0.00	0.00	0.14	1.00	0.00	0.00	0.00	0.00
W	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Y	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
V	0.43	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Fig. 4.2 : Un exemple d'alignement de largeur $W = 10$ et composé de $N = 7$ occurrences, ainsi que la matrice correspondante F, de fréquences des K symboles (acides aminés).

$$C = \begin{pmatrix} c_{1,1} & c_{2,1} & \cdots & c_{W,1} \\ c_{1,2} & c_{2,2} & \cdots & c_{W,2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1,K} & c_{2,K} & \cdots & c_{W,K} \end{pmatrix}$$

où $c_{i,j}$ donne le décompte du symbole σ_j dans la i^{eme} colonne de l'alignement. Nous pourrons cependant par commodité utiliser la notation $c_{i,\sigma}$ pour parler directement du décompte d'un symbole σ dans la i^{eme} colonne. Les fréquences relatives sont représentées dans une matrice $F = C/N$ (avec N étant le nombre d'occurrences dans l'ULMA), $f_{i,j}$ étant la fréquence relative du symbole σ_j dans la i^{eme} colonne, et F_i correspond à la distribution des symboles dans la i^eme colonne. Afin d'illustrer ces propos, la Figure 4.2 présente un exemple d'ULMA de largeur W=10 et composé de $N = 7$ occurrences $\{A_1, A_2, \ldots, A_7\}$ ainsi que la matrice de fréquences F correspondant.

Soit $OF^{(H)}$ la fonction objectif d'entropie relative, avec $OF^{(H)}(C)$ étant le score (ou la fitness) d'un ULMA dont les décomptes C sont issus. Bien que $OF^{(H)}$ ne soit pas

directement fonction de V mais de la matrice des décomptes C de l'alignement, nous utiliserons la notation $OF^{(H)}(V)$ pour parler directement de la fonction objectif d'un alignement[1]. Le vecteur F^0 n'est pas considéré comme une variable de la fonction objectif car il est gardé constant pendant toute la durée de l'optimisation.

$OF^{(H)}(V)$ est définie comme étant la somme sur les colonnes i des entropies relatives $H(F_i|F^0)$ des distributions des symboles :

$$OF^{(H)}(V) = \sum_{i=1}^{W} \sum_{j=1}^{K} f_{i,j} \log_2 \left(\frac{f_{i,j}}{f_j^0} \right) \tag{4.2}$$

Ainsi, $OF^{(H)}(V)$ est maximisée pour l'alignement dont la distribution des symboles dans ses colonnes sera maximalement divergente par rapport à la distribution "bruit de fond".

Il faut mentionner que cette mesure est dans ce contexte équivalente à maximiser au logarithme du ratio des vraisemblances de F et F^0 (Bailey, 1993). En effet, on suppose que la distribution des symboles dans une colonne i suit une loi multinomiale de paramètres F_i. Les colonnes de l'alignement étant supposées indépendantes, on exprime alors la probabilité de l'alignement (ou des décomptes observés dans l'alignement) en fonction des modèles de multinomiale $F_1, F_2, \ldots, F_W$ par :

$$P\left(C|F\right) = \prod_{i=1}^{W} Z(C_i) \prod_{j=1}^{K} \left(f_{i,j}\right)^{c_{i,j}} \tag{4.3}$$

où $Z(C_i)$ correspond au nombre de permutations possibles dans la i^{eme} colonne :

$$Z(C_i) = \frac{N!}{\prod_{j=1}^{K} \left(c_{i,j}!\right)} \tag{4.4}$$

De la même manière, la probabilité de l'alignement par rapport au bruit de fond est exprimé par :

$$P\left(C|F^0\right) = \prod_{i=1}^{W} Z(C_i) \prod_{j=1}^{K} \left(f_j^0\right)^{c_{i,j}} \tag{4.5}$$

Passer des probabilités aux *vraisemblances* n'est qu'un problème de définition : on parle de vraisemblance $\mathcal{L}$ quand on regarde une probabilité comme étant fonction des paramètres F et non plus des données C. Ainsi $P(C|F) = \mathcal{L}(F|C)$. Étant donné que l'on cherche à maximiser $P(C|F)$ et à minimiser $P(C|F^0)$, on va utiliser le ratio des vraisemblances :

$$\frac{\mathcal{L}(F|C)}{\mathcal{L}(F^0|C)} = \prod_{i=1}^{W} \prod_{j=1}^{K} \left(\frac{f_{i,j}}{f_j^0} \right)^{c_{i,j}}$$

Ce qui nous conduit naturellement au logarithme du ratio des vraisemblance LLR (Log Likelihood Ratio) :

[1]La nature "boite noire" de cette fonction par rapport à V apparait clairement. Il n'existe pas de fonction permettant de calculer C directement à partir des positions V, ces valeurs dépendant de l'ensemble de séquences S utilisé.

$$LLR = \log\left(\frac{\mathcal{L}(F|C)}{\mathcal{L}(F^0|C)}\right) = \sum_{i=1}^{W}\sum_{j=1}^{K} c_{i,j}\log\left(\frac{f_{i,j}}{f_j^0}\right) \tag{4.6}$$

Il est alors facile de voir que $OF^{(H)} = (1/N)LLR$.

4.5 L'entropie relative recouvrante

L'entropie relative, bien qu'étant souvent utilisée avec succès souffre d'un problème sérieux lorsqu'il s'agit de mesurer la conservation d'un alignement de protéines (Valdar, 2002). Ce problème concerne la nature des acides aminés. En effet, pour attribuer un score à une colonne, l'entropie relative ne dépend que du décompte des symboles, ainsi que leurs probabilités *a priori*. Chaque symbole est donc traité comme un élément complètement distinct et indépendant des autres. Ainsi, si nous supposons une distribution du bruit de fond F^0 équiprobable (pour simplifier), une colonne contenant 50% de leucines et 50% d'aspartates obtiendra exactement le même score qu'une colonne contenant 50% de leucines et 50% d'isoleucines. Or, tout expert en alignement dirait immédiatement que le deuxième cas (leucine et isoleucine) est biologiquement pertinent et devrait par conséquent recevoir un score supérieur au premier cas. En effet, la leucine et l'isoleucine partagent des caractéristiques communes, rendant la substitution de l'une par l'autre souvent possible sans pour autant affecter les propriétés chimiques de la protéine en question, alors que de substituer une leucine par un aspartate risque fort d'avoir des conséquences quant à la fonctionnalité de la protéine.

Pour illustrer les conséquences de l'utilisation de l'entropie relative sur un alignement de protéines, examinons l'alignement fictif proposé dans le tableau 4.1. Cet alignement montre différentes colonnes présentant différents niveaux de conservation ou de pertinence biologique. Chacune de ces colonnes reçoit un score d'entropie relative $OF^{(H)}$ en utilisant pour F^0 les fréquences relatives des acides aminés observés dans SWISS-PROT (Boeckmann et al., 2003). Les colonnes sont triées de gauche à droite par ordre de score décroissant. Ainsi, la colonne jugée la plus pertinente se situe à gauche et la moins pertinente à droite. Cet ordonnancement de colonnes apparaîtra immédiatement problématique à un expert car des colonnes jugées peu pertinentes sont clairement sur-évaluées par l'entropie relative, alors que d'autres colonnes présentant une bonne pertinence biologique sont sous-évaluées.

Une première approche simple pour pallier à ce problème consisterait à changer d'alphabet en créant des classes exclusives pour les acides aminés similaires. Une telle approche a par exemple été utilisée en regroupant les acides aminés de cette manière (Mirny and Shakhnovich, 1999) : aliphatiques [AVLIMC], aromatiques [FWYH], polaires [STNQ], positifs [KR], négatifs [DE], et 'conformation spéciales' [GP]. Ce procédé est toutefois un peu brutal et réduit considérablement l'information présente dans l'ensemble de séquences. Nous proposons donc de modifier le score d'entropie relative, pour lui permettre d'intégrer une information de *recouvrement* entre les acides aminés. Ce nouveau score, l'*entropie relative recouvrante*, noté $OF^{(HR)}$, permet de prendre en considération une mesure d'équivalence ou de recouvrement entre les acides aminés alignés, tout en gardant les propriétés intéressantes de l'entropie relative.

a	b	c	d	e	f	g	h	i
D	I	D	D	D	I	K	I	D
D	I	D	D	D	I	K	I	V
D	I	D	D	D	I	K	I	Y
D	I	D	D	D	I	K	I	A
D	I	D	D	D	I	D	L	T
D	I	D	V	E	L	D	L	K
D	I	D	V	E	L	D	L	P
D	I	E	V	E	L	G	L	C
D	I	E	V	F	L	G	V	R
D	I	E	V	F	L	G	V	H
4.24	4.08	3.26	3.07	2.74	2.73	2.49	2.24	1.19

Tab. 4.1 : Mesure de conservation par l'entropie relative $OF^{(H)}$. Neuf différentes colonnes d'acides aminés sont évaluées par l'entropie relative en utilisant comme bruit de fond les fréquences d'acides aminés dans SWISS-PROT. Les colonnes sont triées par ordre décroissant de gauche à droite. Les deux premières colonnes sont complètement conservées. La colonne (a) obtient un score supérieur à la colonne (b) car l'aspartate "D" est moins fréquent que l'isoleucine "I". Cependant, des problèmes se posent avec les colonnes (d),(e),(f) et (h). Les colonnes (f) et (h) obtiennent un mauvais score avec l'entropie relative car elle sont composées de (f) deux symboles différents, et (h) trois symboles différents. De plus, ces acides aminés "I","L" et "V" sont relativement fréquents dans SWISS-PROT. Cependant, ces trois acides aminés partagent plusieurs caractéristiques physico-chimiques qui les rendent facilement substituables dans une protéine. On peut donc dire que ces colonnes sont sous-évaluées par l'entropie relative. Réciproquement, les colonnes (d) et (e) sont quant à elles sur-évaluées car les acides-aminés "D" et "V" ne sont pas facilement substituables dans une protéine sans en affecter les propriétés. Cette observation est également vraie pour les acides aminés "D" et "F" ou "E" et "F".

4.5.1 Une mesure de recouvrement

Le principe de l'entropie relative recouvrante est de pouvoir utiliser une mesure de recouvrement entre les symboles pris deux à deux. Par recouvrement, nous entendons une "équivalence" partielle bien que le terme d'*équivalence* peut paraître mal approprié, car dans son utilisation courante, une équivalence peut être soit vraie soit fausse. Supposons qu'un symbole x soit défini comme recouvrant un symbole y à 50%, cela signifie que l'observation d'un symbole x est équivalente à l'observation d'une moitié de symbole y, et donc que l'observation de deux symboles x équivaut à l'observation d'un symbole y. Cette valeur de recouvrement doit être définie pour toutes les paires de symboles : soit M une matrice où $M(x, y)$ est la valeur de recouvrement entre les symboles x et y. Cette mesure peut prendre des valeurs réelles comprises entre 0.0 et 1.0, une valeur de 0.0 signifiant que le recouvrement est nul, alors qu'une valeur de 1.0 signifie que le recouvrement est maximal. Ainsi, si $M(x, y) = 1.0$, l'observation de x équivaut à l'observation de y. Par définition, pour tout x, $M(x, x) = 1.0$, car un symbole est toujours maximalement recouvrant avec lui-même. Pour des raisons de simplicité, nous supposons cette matrice comme étant symétrique : $\forall x, y\ M(x, y) = M(y, x)$.

En l'absence d'informations adéquates pour déterminer ces valeurs de recouvrement, nous avons opté pour une solution simple, (ayant cependant donné d'excellents résultats), en nous basant sur les données brutes qui ont servis à l'élaboration des matrices de substitution BLOSUM (Henikoff and Henikoff, 1992). Ces matrices sont construites à partir de fréquences de couples d'acides aminés, observés dans des alignements multiples locaux et sans gaps contenus dans la base de données BLOCKS (Henikoff and Henikoff, 1991; Henikoff et al., 2000). Cette base de données a été partitionnée pour regrouper les alignements (blocs) contenant des séquences de même niveau de similarité. Par exemple, la matrice BLOSUM62 est construite à partir de fréquences de couples d'acides aminés observés dans des blocs présentant une similarité d'au moins 62%. Les entrées $B_{i,j}$ d'une matrice BLOSUM sont calculées de la façon suivante :

$$B_{i,j} = 2 \log_2 \left(\frac{q_{i,j}}{f_{i,j}} \right)$$

où $q_{i,j}$ est la fréquence observée de couples formés par les acides aminés i et j, alors que $q_{i,j}$ est la fréquence que l'on attendrait par hasard. L'utilisation du logarithme permet d'avoir un score additif (lequel est directement utilisé par les méthodes d'alignement deux à deux par programmation dynamique (Section 3.3)). La multiplication par deux est arbitraire et produit une unité de demi bits.

Soit $g_{i,j} = q_{i,j}/f_{i,j}$, le ratio des fréquences observées sur les fréquences attendues. Nous calculons simplement une matrice M de similarité normalisée à 1 de la façon suivante :

$$M_{i,j} = \frac{g_{i,j}}{\left(g_{i,i} + g_{j,j}\right)/2}$$

Le résultat d'une telle matrice calculée à partir des fréquences de couples correspondant à une BLOSUM62 est présenté à la figure 4.3. Cette façon de procéder, bien qu'un peu brutale, a produit des valeurs jugées acceptables par plusieurs experts et a permis d'obtenir des résultats tout à fait pertinents. Comme nous l'avons déjà dit, la propriété de symétrie de la matrice ne se justifie pas a priori, mais en l'absence d'une théorie plus complète pour calculer ces valeurs, il serait déraisonnable de compliquer inutilement le modèle. Il va sans

dire que la détermination d'une ou de plusieurs matrices plus appropriées ou spécifiques à une problématique donnée reste ouverte.

4.5.2 Intégration des valeurs de recouvrement à l'entropie relative

Nous allons à présent décrire une façon d'intégrer les mesures de recouvrement au score de l'entropie relative. Comme pour l'entropie relative, F représente la matrice de fréquences des symboles dans l'ULMA, et F^0 est le vecteur des probabilités du bruit de fond. Soit $C^{\oplus}$ une matrice de *décomptes recouvrants*, laquelle a la même taille que C, $c^{\oplus}_{i,j}$ étant le décompte recouvrant du symbole j dans la i^{eme} colonne de l'alignement. Les valeurs de $C^{\oplus}$ sont définies pour tous $i \in \{1..W\}$ et pour tous $j \in \{1..K\}$ par :

$$c^{\oplus}_{i,j} = \sum_{k=1}^{K} C_{i,k} M(j,k) \tag{4.7}$$

Une matrice de *fréquences recouvrante* $R = C^{\oplus}/N$ est obtenue par normalisation sur N, le nombre d'occurrences. Il faut noter que contrairement à F, la matrice R ne peut plus être considérée comme une matrice de probabilité. En effet, dans le cas de F, chaque colonne F_i correspond à une distribution de probabilités, et $\sum_{j=1}^{K} f_{i,j} = 1.0$ pour tous i. Cette propriété n'est plus satisfaite avec R_i pour laquelle la somme des valeurs d'une colonne est supérieure à 1.0. Ce comportement est produit par le recouvrement qui permet à un symbole d'être compté plusieurs fois. Le terme de fréquence recouvrante semble donc approprié.

De façon similaire, nous définissons un vecteur R^0 où r^0_i donne la fréquence recouvrante "bruit de fond" du symbole i :

$$r^0_i = \sum_{k=1}^{K} f^0_k M(i,k) \tag{4.8}$$

La nouvelle fonction objectif $OF^{(HR)}$ est définie par :

$$OF^{(HR)}(V) = \sum_{i=1}^{W} \sum_{j=1}^{K} f_{i,j} \log\left(\frac{r_{i,j}}{r^0_j}\right) \tag{4.9}$$

Si nous nous référons à la théorie de l'information et en particulier à l'équation (4.1) cette fonction peut être réécrite de la façon suivante :

$$OF^{(HR)}(R|R^0) = \mathbb{E}_F\left(I_{R^0} - I_R\right) \tag{4.10}$$

Ce qui correspond à l'espérance sous F de l'information relative entre R et R^0.

Afin de décrire de façon visuelle le comportement de cette fonction, nous proposons dans la Figure 4.4 un graphique représentant les valeurs de $OF^{(HR)}$ en fonction de la variation de la distribution de deux symboles dans une colonne, ainsi que de la valeur de recouvrement entre ces deux symboles.
La Table 4.2 présente les améliorations apportées par cette fonction objectif sur l'ordonnancement des colonnes de l'alignement présenté auparavant dans la Table 4.1.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	1.00																			
R	0.15	1.00																		
N	0.08	0.15	1.00																	
D	0.08	0.07	0.18	1.00																
C	0.08	0.03	0.02	0.02	1.00															
Q	0.15	0.25	0.15	0.15	0.03	1.00														
E	0.15	0.18	0.15	0.29	0.02	0.35	1.00													
G	0.17	0.07	0.12	0.09	0.02	0.07	0.07	1.00												
H	0.05	0.09	0.12	0.06	0.02	0.09	0.09	0.04	1.00											
I	0.18	0.07	0.06	0.06	0.05	0.07	0.07	0.04	0.04	1.00										
L	0.18	0.10	0.06	0.04	0.05	0.10	0.07	0.04	0.04	0.50	1.00									
K	0.15	0.35	0.15	0.10	0.03	0.25	0.25	0.07	0.07	0.07	0.10	1.00								
M	0.15	0.12	0.07	0.05	0.05	0.18	0.09	0.05	0.05	0.29	0.41	0.12	1.00							
F	0.08	0.05	0.04	0.04	0.03	0.05	0.05	0.04	0.06	0.17	0.17	0.05	0.15	1.00						
P	0.09	0.06	0.05	0.07	0.02	0.08	0.08	0.05	0.04	0.05	0.05	0.08	0.06	0.03	1.00					
S	0.35	0.15	0.24	0.17	0.05	0.21	0.21	0.17	0.07	0.12	0.12	0.21	0.15	0.08	0.09	1.00				
T	0.21	0.12	0.15	0.10	0.05	0.12	0.12	0.07	0.05	0.15	0.15	0.12	0.12	0.07	0.08	0.29	1.00			
W	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.02	0.02	0.01	0.02	0.01	0.03	0.05	0.01	0.01	0.02	1.00		
Y	0.07	0.06	0.05	0.04	0.03	0.08	0.06	0.04	0.15	0.09	0.09	0.06	0.08	0.29	0.03	0.07	0.06	0.07	1.00	
V	0.25	0.07	0.06	0.06	0.05	0.10	0.10	0.06	0.04	0.71	0.35	0.10	0.29	0.12	0.07	0.12	0.21	0.01	0.09	1.00

Fig. 4.3 : Une matrice de recouvrement calculée à partir des fréquences de couples d'acides aminés observés dans la base de données BLOCKS. Chaque acide aminé contre lui-même obtient une valeur de recouvrement maximale de 1.0 .

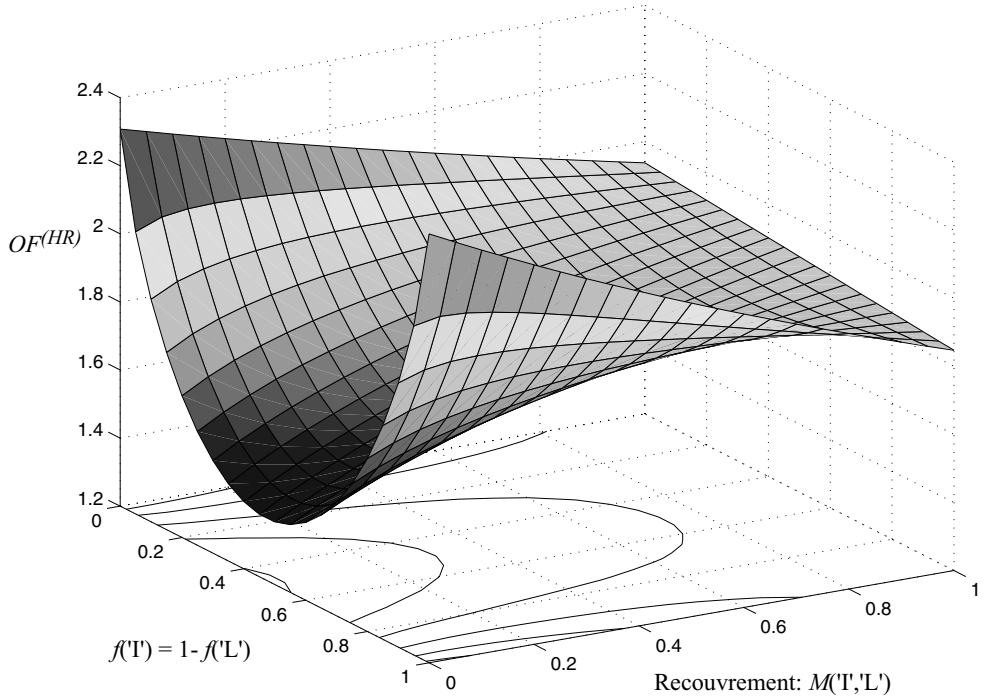


Fig. 4.4 : Ce graphique représente les valeurs de l'entropie relative recouvrante en fonction de la distribution de deux acides aminés : isoleucine (I) et leucine (L), ainsi qu'en fonction de leur valeur de recouvrement M('I','L') variant entre 0 et 1. Nous avons assigné à F^0 les fréquences observées des acides aminés dans SWISS-PROT (Boeckmann et al., 2003). Les mêmes valeurs de recouvrement que celles présentées dans la Figure 4.3 sont utilisées pour calculer R^0 (à l'exception de M('I','L') qui varie). Lorsque M('I','L') = 0.0, la courbe ressemble fortement à ce que l'on obtiendrait avec l'entropie relative "classique" $OF^{(H)}$. Une petite différence est cependant due au fait que les autres acides aminés partagent également des comptes et donc $R^0 \neq F^0$. Plus M('I','L') augmente, et moins la distribution des deux symboles n'a d'influence sur $OF^{(HR)}$. Quand M('I','L') vaut 1.0 la variation d'entropie recouvrante devient linéaire, avec une valeur légèrement plus grande pour le symbole dont le bruit de fond recouvrant est le plus bas. Il faut également noter que les valeurs de $OF^{(HR)}$ pour M('I','L') = 1.0 sont légèrement plus basses que lorsque les deux acides aminés ne présentent pas de recouvrement. Cette diminution est due au fait que les valeurs de R^0 augmentent avec le recouvrement rendant l'apparition des acides aminés concernés plus probables a priori.

	a	b	c	d	e	f	g	h	i
	D	I	D	D	D	I	K	I	D
	D	I	D	D	D	I	K	I	V
	D	I	D	D	D	I	K	I	Y
	D	I	D	D	D	I	K	I	A
	D	I	D	D	D	I	D	L	T
	D	I	D	V	E	L	D	L	K
	D	I	D	V	E	L	D	L	P
	D	I	E	V	E	L	G	L	C
	D	I	E	V	F	L	G	V	R
	D	I	E	V	F	L	G	V	H
OF^{H}	4.24	4.08	3.26	3.07	2.74	2.73	2.49	2.24	1.19

	a	b	c	f	h	e	d	g	i
	D	I	D	I	I	D	D	K	D
	D	I	D	I	I	D	D	K	V
	D	I	D	I	I	D	D	K	Y
	D	I	D	I	I	D	D	K	A
	D	I	D	I	L	D	D	D	T
	D	I	D	L	L	E	V	D	K
	D	I	D	L	L	E	V	D	P
	D	I	E	L	L	E	V	G	C
	D	I	E	L	V	F	V	G	R
	D	I	E	L	V	F	V	G	H
$OF^{(HR)}$	2.77	2.12	2.11	1.70	1.57	1.57	1.52	1.30	0.36

Tab. 4.2 : Différence d'ordonnancement de colonnes de symboles avec l'entropie relative $OF^{(H)}$ (tableau du haut) et l'entropie relative recouvrante $OF^{(HR)}$ (tableau du bas). Le tableau du haut est le même que celui de la Table 4.1 et est présenté pour faciliter la comparaison. Le tableau du bas reprend les mêmes colonnes, mais en les réordonnant selon leur score d'entropie relative recouvrante. Nous pouvons constater que les sur-évaluations et sous-évaluations constatées avec la fonction $OF^{(H)}$ sont corrigées avec la fonction $OF^{(HR)}$. En particulier, les colonnes (f) et (h) obtiennent une position dans l'ordonnancement sous $OF^{(HR)}$ qui semble acceptable. Il en va de même pour les colonnes (d) et (e) qui étaient sur-évaluées par $OF^{(H)}$.

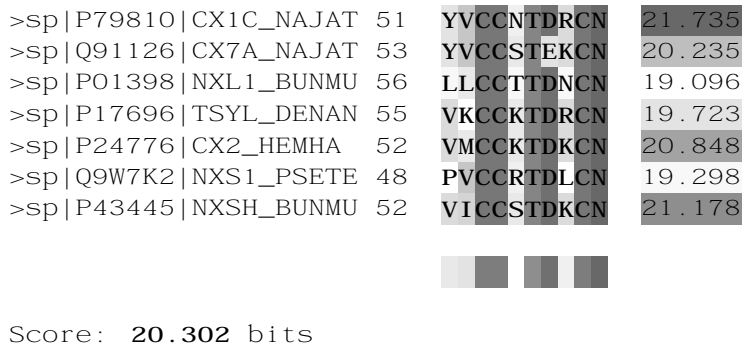


Fig. 4.5 : Un alignement d'un motif caractéristique de toxines de serpent riche en cystéines ('C') est représenté. Chaque occurrence est précédée par un label correspondant à l'entrée de la séquence concernée dans SWISS-PROT. Le numéro qui suit est la position de l'occurrence dans la séquence. Chaque symbole est représenté sur un fond plus ou moins foncé mettant en évidence sa conservation dans l'alignement. La valeur venant ensuite est le score de l'occurrence, lequel est calculé pour une occurrence A_i par $\sum_{j=1}^{W} \log_2(r_{j,A_{i,j}}/r^0_{A_{i,j}})$. La moyenne de ces scores d'occurrence redonne la valeur de la fonction objectif pour l'alignement. Finalement sont représentés les scores de chaque colonne.

4.5.3 Représentation d'un alignement

Il est important de pouvoir représenter un alignement de façon lisible, en mettant en évidence les parties conservées. Nous proposons une méthode consistant à colorer plus ou moins chaque symbole de l'alignement de façon proportionelle à sa "conservation" dans la colonne. Chaque symbole $A_{i,j}$ est plus ou moins coloré en fonction de :

$$\text{Coloration}(A_{i,j}) \div r_{i,j} \log\left(\frac{r_{i,j}}{r^0_j}\right)$$

(Avec la notation '$\div$' signifiant "est proportionnel à").
En utilisant le même alignement que celui présenté à la Figure 4.2, nous présentons sa représentation telle qu'elle est produite par notre implémentation en utilisant l'entropie relative recouvrante (Figure 4.5). Cet alignement a été obtenu sur un petit ensemble de séquences protéiques de toxines de serpents (nous remercions Stefanie Kappus et Reto Stoecklin qui nous ont fourni ces données), en utilisant l'entropie relative recouvrante comme fonction objectif (les stratégies d'optimisation sont décrites plus loin dans ce chapitre). Cet alignement met en évidence une région riche en cystéines ('C') caractéristiques de cette classe de toxines.

4.6 Stratégies pour l'optimisation d'alignements

Les précédentes sections ont décrit le problème général de l'alignement local multiple sans indels (ULMA), ainsi que deux fonctions objectif permettant de mesurer la conservation des occurrences. Nous présentons dans cette section une approche originale pour la découverte d'ULMA maximisant une des deux fonctions objectif. Cette approche repose sur une modélisation du problème sous la forme d'un problème d'optimisation combinatoire. En premier lieu, nous décrirons quatre modèles de contrainte pour la répartition des

occurrences dans l'ensemble de séquences. Ils permettent de représenter quatre situations classiques auxquelles sont confrontés les biologistes lorsqu'ils cherchent à découvrir de nouveaux motifs. Nous décrirons ensuite un grimpeur strict fonctionnant en alternance sur deux voisinages spécifiques au problème. Nous décrirons alors trois stratégies qui portent sur le choix des *graines* à optimiser. Une graine dénomme le point de l'espace de recherche (un ULMA) qui sert de départ au grimpeur.

4.6.1 Quatre modèles de contrainte pour la répartition des occurrences

Nous définissons quatre modes (des paramètres utilisateur), correspondant chacun à une façon de contraindre la répartition des N occurrences de l'ULMA dans l'ensemble de séquences. Ces contraintes permettent à l'utilisateur de spécifier un a priori qu'il peut avoir sur cette répartition et réduire de ce fait la taille de l'espace de recherche. Il serait par exemple dommage de chercher à optimiser un alignement en autorisant une disposition libre des occurrences sur les séquences, si l'utilisateur *sait* ou *suppose* que chaque séquence doit contenir exactement une occurrence de l'ULMA. Cette information permet de réduire considérablement la taille de l'espace explorable. De plus, une contrainte plus forte aura un effet considérable sur le niveau de bruit. En effet, plus on apporte d'information sur ce que l'on cherche et plus on réduit le niveau du bruit. Il est donc important dans la mesure du possible d'utiliser la contrainte la plus forte possible en fonction des connaissances préalables du biologiste. Dans le cas d'une similarité faible par exemple, une contrainte plus forte peut être suffisante pour extraire le signal du bruit.
Avant de commencer la description des ces quatre modes, rappelons rapidement les notations définies précédemment (Section 4.2.2). Un ULMA est codé par un vecteur $V = (v_1, v_2, \ldots, v_N)$, où chaque $v_i \in \{1, 2, \ldots, T(L - W + 1)\}$ représente un élément de l'ensemble S^* des facteurs de longueur W que l'on peut constituer à partir de l'ensemble de séquences S. Les T séquences de S sont supposées avoir toutes la même taille L. La fonction $\eta(v_i) = j$ signifie que le facteur représenté par v_i se trouve dans S_j, la j^{eme} séquence de S.

Le mode OOPS : définitions et notations

Une première façon de contraindre la disposition des occurrences dans l'ensemble de séquences est de forcer chaque séquence à contribuer exactement une fois à l'alignement. Cette contrainte est la plus forte des quatre et certainement la plus utilisée. Nous appelons cette contrainte le mode 'OOPS' (One Occurrence Per Sequence) (Figure 4.6a) en référence au programme MEME (Bailey and Elkan, 1995). Ce mode est approprié si l'on suppose que toutes les séquences de S sont apparentées et qu'elles contiennent donc toutes une (ou au moins une) occurrence du motif que l'on cherche à découvrir.
L'ensemble des solutions réalisables sous la contrainte OOPS peut être implicitement redéfini en restreignant le domaine des variables $V = (v_1, v_2, \ldots, v_N)$ de façon à ce que chaque v_i ne puisse plus représenter que des occurrences se trouvant sur la i^{eme} séquence :

$$v_i \in \{(i-1)(L-W+1)+1, \ldots, i(L-W+1)\}$$

En utilisant le mode OOPS, le nombre d'occurrences N est implicitement défini par T le nombre de séquences que l'on cherche à aligner.

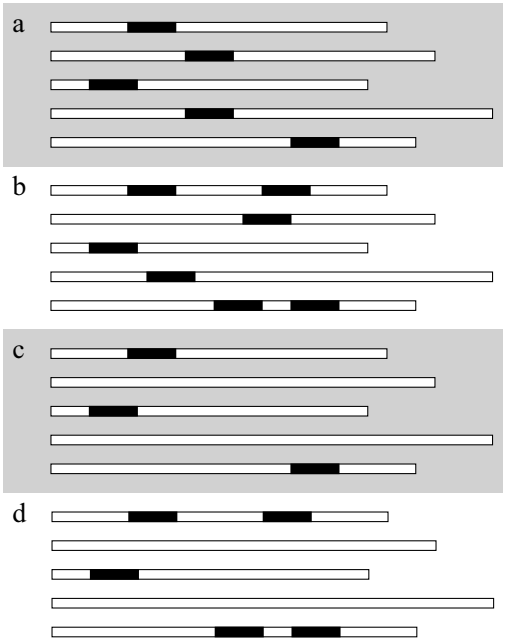


Fig. 4.6 : Cette Figure représente un exemple d'ULMA pour chacun des quatre modèles de contrainte pour la répartition des occurrences de l'ULMA dans les séquences. a) Exactement une occurrence par séquence. b) Au moins une occurrence par séquence. c) Au plus une occurrence par séquence. d) zéro, une ou plusieurs occurrences par séquence.

Le mode ALOOPS : définitions et notations

Une deuxième façon de contraindre la disposition des occurrences dans l'ensemble de séquences consiste à forcer chaque séquence à contribuer au moins une fois à l'alignement (Figure 4.6b). Ce mode appelé 'ALOOPS' (At Least One Occurence Per Sequence) est légèrement moins contraint que le premier, mais suppose toujours que toutes les séquences doivent participer à l'alignement. Dans ce cas, $T \leq N \leq MaxOcc$, avec $MaxOcc = \lfloor T(L - W + 1)/(W) \rfloor$ correspondant au nombre maximum d'occurrences non recouvrantes. Il est clair qu'en pratique, $N << MaxOcc$.

A partir du moment où l'on autorise plusieurs occurrences d'un ULMA à se positionner sur la même séquence, il est important de s'assurer qu'elles n'occupent pas la même position, c'est-à-dire qu'elles ne se chevauchent pas. Si cette mesure n'était pas utilisée, l'optimisation convergerait vers un ULMA composé du même mot (dans le cas d'un recouvrement complet), ou de mots situés sur une région de basse complexité (répétitions dans les séquences). Soit $Ov(v_i, v_j) \in \{0, 1\}$ une fonction booléenne définie par :

$$Ov(v_i, v_j) = \begin{cases} 0 & \text{si } |v_i - v_j| \geq W \\ 0 & \text{si } \eta(v_i) \neq \eta(v_j) \\ 1 & \text{sinon} \end{cases}$$

De cette façon, $Ov(v_i, v_j) = 0$, si et seulement si les facteurs dénotés par v_i et v_j ne se chevauchent pas.

Une solution réalisable est un vecteur $V = (v_1, v_2, \ldots, v_N)$ avec $N \geq L$, et :

$$\forall i \in \{1..T\},\ \exists j \in \{1..N\} \mid \eta(v_j) = i$$
et
$$\forall i\ \in \{1..N\} \forall j \in \{1..N\}\ i \neq j,\ \ Ov(v_i, v_j) = 0$$

Le mode AMOOPS : définitions et notations

Le troisième mode 'AMOOPS' (At Most One Occurrence Per Sequence) (Figure 4.6c) est utile lorsque l'utilisateur suppose que le motif qu'il recherche n'est présent qu'une fois par séquence, mais il n'est pas sûr que toutes les séquences de son ensemble contiennent effectivement le motif recherché (ses données sont polluées par des séquences non-apparentées). Ce mode permet d'ignorer un nombre donné de séquences de l'alignement final. Une solution $V = (v_1, v_2, \ldots, v_N)$ où $N \leq T$ et est réalisable si :

$$\forall i \in \{1..N\} \forall j \in \{1..N\}\ \ i \neq j \Leftrightarrow \eta(v_i) \neq \eta(v_j)$$

Le mode AOPS : définitions et notations

Le quatrième et dernier mode 'AOPS' pour "Any number of Occurrence Per Sequence" (Figure 4.6d) permet aux occurrences d'être réparties n'importe où dans l'ensemble de séquences, avec la seule contrainte qu'elles ne se recouvrent pas. Ce mode est le moins contraint des quatre. Une solution $V = (v_1, v_2, \ldots, v_N)$ est réalisable si :

$$\forall i\ \in \{1..N\} \forall j \in \{1..N\}\ i \neq j,\ \ Ov(v_i, v_j) = 0$$

La taille des ensembles de solutions

On peut approximer pour chacun des quatre modes la taille de l'ensemble des solutions réalisables. Soit $\mathcal{P} = \{V_1, V_2, \ldots, V_Z\}$ l'ensemble des solutions réalisables pour un mode donné, avec Z étant la taille de l'ensemble. D'une façon générale, la taille de l'ensemble est exponentielle sur N et peut être approximée par $\widetilde{U}^N$, $\widetilde{U}$ donnant le nombre moyen de valeurs que les v_i peuvent prendre. Pour le mode OOPS, le taille du domaine des v_i est parfaitement définie par $\widetilde{U} = L - W + 1$. Concernant le mode ALOOPS, le nombre de valeurs que peut prendre un v_i peut fortement varier en fonction de la valeur des autres variables (répartition courante des autres occurrences). On peut cependant approximer la valeur moyenne par $\widetilde{U} = T(L - W + 1)(1 + N - T)/N$. Pour le mode AMOOPS, $\widetilde{U}$ vaut exactement $(T - N + 1)(L - W + 1)$. Finalement, pour le mode AOPS, $\widetilde{U}$ est approximé par $T(L - W + 1) - NW$.

Fonctions de voisinage

Soit $\mathcal{N}^{OOM}(V^*, i)$ la fonction de voisinage "One Occurrence Move", avec V^* un élément de l'ensemble des solutions réalisables $\mathcal{P}$ pour le mode utilisé, et i dénote la i^{eme} dimension de V^* (v_i^* avec $i \in \{1, 2, \ldots, N\}$). $\mathcal{N}^{OOM}(V^*, i)$ produit l'ensemble des solutions $\{V_1, V_2, \ldots, V_U\}$ (incluant V^*) correspondant à toutes les instanciations réalisables de v_i^* (pour le mode concerné), les autres dimensions (occurrences) restant inchangées. La taille du voisinage U sera fonction de la contrainte sur la répartition des occurrences que l'on utilise, ainsi que de la disposition courante des autres occurrences. La taille moyenne est

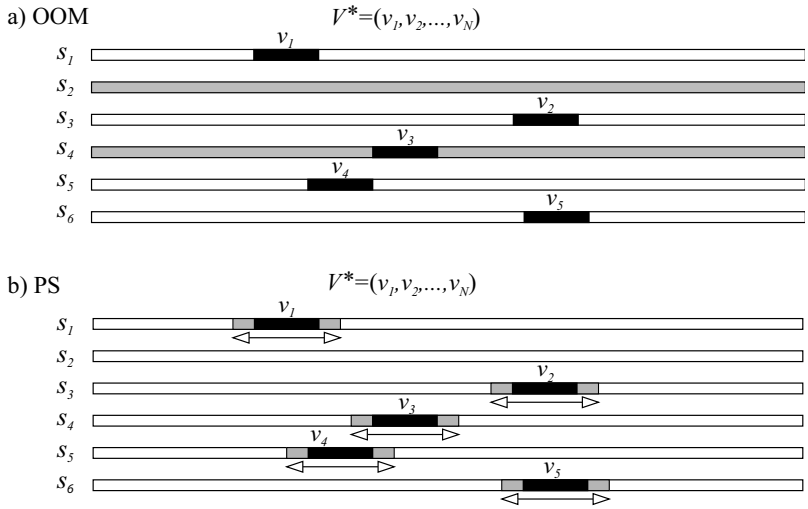


Fig. 4.7 : Deux fonctions de voisinage sont utilisées par les grimpeurs stricts. A) $\mathcal{N}^{OOM}(V^*, i)$ le voisinage "One Occurrence Move" correspond à toutes les instanciations réalisables de la variable v_i, ce qui correspond à toutes les façons de positionner la i^{eme} occurrence selon la contrainte spécifiée. Dans l'exemple représenté, le mode AMOOPS est utilisé et l'occurrence v_3 ne peut donc être positionnée que sur les séquences s_2 et s_4. B) $\mathcal{N}^{PS}(V^*)$ le voisinage "Phase Shift" comprend tous les ULMAs obtenus par un décalage simultané de toutes les occurrences en même temps. Le décalage maximum est fixé à plus ou moins la moitié de la largeur de l'alignement. Cette opération est très importante pour l'efficacité des grimpeurs car elle permet d'assurer que l'alignement ne se trouve pas dans un maximum local correspondant à un décalage simultané de ses occurrences.

cependant approximée par $\widetilde{U}$ qui est défini dans la Section 4.6.1.

Soit $\mathcal{N}^{PS}(V^*)$ le voisinage "Phase Shift". Ce voisinage produit l'ensemble des solutions réalisables définies par $V^* + \delta$ avec :

$$\delta \in \{-\lfloor (W+1)/2 \rfloor, \ldots, 0, \ldots, \lfloor (W+1)/2 \rfloor\}$$

L'ensemble des ULMAs correspondant à un décalage simultané de *toutes* les occurrences sur plus ou moins la moitié de la largeur sont produits. Ce voisinage est extrêmement important pour corriger un mauvais "calage" des occurrences que la première fonction de voisinage serait incapable de faire, et améliore de ce fait beaucoup la performance d'une optimisation s'il est utilisé. Le choix d'un décalage sur la moitié de la largeur de l'alignement est empirique, mais largement suffisant pour corriger un mauvais "calage" des occurrences.
La figure 4.7 illustre graphiquement ces deux fonctions de voisinage.

Complexité temporelle de l'évaluation du voisinage $\mathcal{N}^{OOM}$ sous $OF^{(H)}$

La complexité temporelle de l'évaluation de l'ensemble des voisins produits par $\mathcal{N}^{OOM}(V^*, i)$, en utilisant l'entropie relative comme fonction objectif est de : $\mathcal{O}(UW)$, (avec U pour la taille du voisinage, et W pour la largeur de l'ULMA V^*). En effet, calculer $OF^H(V^*)$ à partir de zéro implique une double somme sur N (le nombre d'occurrences de V^*) et sur W (sa largeur), pour calculer la matrice de décompte C, ainsi qu'une double

somme sur K (le cardinal de l'alphabet) et sur W pour calculer $F^H(C)$, soit une complexité de : $\mathcal{O}(W(N+K))$. Cependant, les ULMAs du voisinage $\mathcal{N}^{OOM}$ ne diffèrent que d'une seule occurrence, les autres restant inchangées. Or mettre à jour le score d'un ULMA pour lequel on a ajouté, retiré ou modifié une seule de ses occurrences se fait linéairement sur W. Commençons par reformuler l'équation 4.2 de l'entropie relative d'un ULMA en fonction des décomptes (et non plus des fréquences relatives) :

$$\begin{aligned} OF^{(H)}(V) &= \sum_{i=1}^{W}\sum_{j=1}^{K}\frac{c_{i,j}}{N}\log\left(\frac{c_{i,j}}{f_j^0 N}\right) \\ &= \frac{1}{N}\left[\sum_{i=1}^{W}\sum_{j=1}^{K} c_{i,j}\log\left(\frac{c_{i,j}}{f_j^0}\right)\right] - W\log(N) \end{aligned} \tag{4.11}$$

Étant donné que W et N sont constants, nous pouvons définir une nouvelle fonction :

$$OF^{(\div H)}(V) = \sum_{i=1}^{W}\sum_{j=1}^{K} c_{i,j}\log\left(\frac{c_{i,j}}{f_j^0}\right) \tag{4.12}$$

laquelle est une transformation linéaire de $OF^{(H)}$:

$$OF^{(\div H)} = N\left(OF^{(H)}(V) + \log(N)W\right)$$

Ces deux fonctions sont donc équivalentes à maximiser par un grimpeur strict, ou par toute méthode ne considérant que l'ordonnancement des valeurs de la fonction.
$OF^{(\div H)}$ correspond à la somme de KW termes indépendants, chacun pouvant être exprimé par une fonction $L(c_{i,j})$ définie par :

$$L(c_{i,j}) = c_{i,j}\log_2\left(\frac{c_{i,j}}{f_j^0}\right) \tag{4.13}$$

(L est uniquement fonction des valeurs $c_{i,j}$ de C car le vecteur de bruit de fond F^0 est constant). $OF^{(\div H)}$ peut donc se réécrire :

$$OF^{\div H}(V) = \sum_{i=1}^{W}\sum_{j=1}^{K} L(c_{i,j}) \tag{4.14}$$

Ajouter ou retirer une occurrence s'effectue en ne modifiant que les W termes concernés dans la somme. Nous ne présentons ici que l'opération consistant à ajouter une occurrence, le retrait s'effectuant selon le même principe. Soit $V^{\ominus}$, un ULMA composé de $N-1$ occurrences $\{A_1, A_2, \ldots, A_{N-1}\}$ et soit $C^{\ominus}$ la matrice de décompte de $V^{\ominus}$. Soit A_N l'occurrence que l'on désire rajouter à $V^{\ominus}$ avec $A_{N,i}$ le i^{eme} symbole de A_N. Le nouvel ULMA obtenu par l'ajout de l'occurrence A_N à $V^{\ominus}$ est noté V'. Le score $OF^{(\div H)}(V')$ est calculé en temps linéaire sur W à partir de $OF^{(\div H)}(V^{\ominus})$ par :

$$OF^{(\div H)}(V') = OF^{(\div H)}(V^{\ominus}) + \sum_{i=1}^{W}\left(L(C^{\ominus}_{i,A_{N,i}}+1) - L(C^{\ominus}_{i,A_{N,i}})\right) \tag{4.15}$$

ce qui donne une complexité de $\mathcal{O}(UW)$ pour évaluer U voisins. $OF^{(H)}(V')$ est ensuite obtenu par simple transformation linéaire de $OF^{(\div H)}(V')$.

Complexité temporelle de l'évaluation du voisinage $\mathcal{N}^{OOM}$ sous $OF^{(HR)}$

La complexité temporelle de l'évaluation du voisinage $\mathcal{N}^{OOM}$ sous la fonction objectif $OF^{(HR)}$ est de $\mathcal{O}(W(U+K^2))$. La différence majeure avec la complexité sous $OF^{(H)}$ est que comme le montre l'équation 4.7, le calcul des décomptes pour une colonne nécessite K^2 opérations car chaque symbole doit être confronté à son recouvrement avec tous les autres symboles.

Similairement à l'équation 4.12 nous pouvons définir $OF^{(\div HR)}$ par une somme de WK termes indépendants :

$$OF^{(\div HR)}(V) = \sum_{i=1}^{W}\sum_{j=1}^{K} L^{(HR)}(c_{i,j}) \tag{4.16}$$

La fonction $L^{(HR)}$ étant définie par :

$$L^{(HR)}(c_{i,j}) = c_{i,j}\log_2\left(\frac{\sum_{k=1}^{K} c_{i,j}M(j,k)}{r_j^0}\right) \tag{4.17}$$

En reprenant l'équation 4.15, le score $OF^{(\div HR)}(V')$ est calculé à partir de $OF^{(\div HR)}(V^{\ominus})$ par :

$$OF^{(\div HR)}(V') = OF^{(\div HR)}(V^{\ominus}) + \sum_{i=1}^{W}\left(L^{(HR)}(C^{\ominus}_{i,A_{N,i}}+1) - L^{(HR)}(C^{\ominus}_{i,A_{N,i}})\right) \tag{4.18}$$

Cette équation nous conduit à une complexité brute de $\mathcal{O}(UWK)$. Cependant, étant donné que $C^{\ominus}$ est constant (les $N-1$ occurrences de $V^{\ominus}$ restent inchangées pour un voisinage donné), il suffit de précalculer $L^{(HR)}$ pour toutes les valeurs de $C^{\ominus}_{i,j}$ et de $C^{\ominus}_{i,j}+1$, soit $2WK$ valeurs. La complexité de ce précalcul est alors donnée par $\mathcal{O}(WK^2)$, ce qui ramène la complexité de l'évaluation de l'équation 4.18 à $\mathcal{O}(UW+WK^2)$ soit $\mathcal{O}(W(U+K^2))$. Cette complexité sera avantageuse par rapport à $\mathcal{O}(UWK)$ pour autant qu'approximativement $U > K$. Dans la grande majorité des applications qui nous intéressent, on a $U >> K$. Par exemple, si l'on considère un alphabet de taille $K = 20$ (protéines), et le mode correspondant au voisinage de plus petite taille (le mode OOPS), U est de l'ordre de grandeur de la taille de la séquence, c'est à dire de quelques centaines d'acides aminés dans la plupart des cas. Il va sans dire que pour les autres modes d'optimisation, l'avantage est encore plus important.

4.6.2 Première stratégie : un grimpeur strict sur des graines aléatoires

La première stratégie consiste à lancer des grimpeurs sur des éléments de $\mathcal{P}$ choisis aléatoirement. Cette stratégie est appelée "la stratégie $\mathcal{P}$". Une graine sera donc un ULMA V dont les instanciations sont choisies aléatoirement de façon à satisfaire les contraintes relatives au mode spécifié. Ce processus d'instanciation aléatoire ne pose pas de problème algorithmique particulier : les variables sont instanciées successivement. A chaque étape, l'ensemble des valeurs possibles pour v_i peut être énuméré, les instanciations précédentes n'ayant jamais besoin d'être remises en cause. Le grimpeur strict utilise deux fonctions de voisinages en alternance, et évolue donc dans deux paysages distincts.

Pseudocode 2 Grimpeur Strict sur $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$

entrée :V^* Un ULMA "graine"
sortie :V' Un ULMA après convergence
$V' = V^*$
FAIRE
  $V = V'$
  $V' = SHC^{PS}(V')$
  POUR TOUS $i \in \{1..N\}$(choisis aléatoirement sans remise)
    $V' = SHC^{OOM}(V', i)$
TANT QUE $OF(V) < OF(V')$

Un grimpeur sur $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$

Les deux fonctions de voisinage $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$ sont utilisées en alternance par un grimpeur strict. Soit V^* un ULMA "graine" tiré au hasard dans l'ensemble P des solutions réalisables pour un mode donné. Le premier opérateur de déplacement $SHC^{OOM}(V^*, i)$ opérant sur le voisinage $\mathcal{N}^{OOM}$ est défini par :

$$\begin{array}{l} SHC^{OOM}(V^*, i) = \\ V_j \in \mathcal{N}^{OOM}(V^*, i) | \forall V_k \in \mathcal{N}^{OOM}(V^*, i), OF(V_j) \geq OF(V_k) \end{array} \tag{4.19}$$

Le deuxième opérateur de déplacement $SHC^{PS}(V^*)$ opère sur le voisinage $\mathcal{N}^{PS}$ et est défini par :

$$\begin{array}{l} SHC^{PS}(V^*) = \\ V_i \in \mathcal{N}^{PS}(V^*) | \forall V_j \in \mathcal{N}^{PS}(V^*), OF(V_i) \geq OF(V_j) \end{array} \tag{4.20}$$

Ces deux opérateurs sont utilisés en alternance par un grimpeur strictement ascendant décrit dans le Pseudocode 2. Ce grimpeur possède une composante stochastique concernant la façon de choisir les occurrences de v_i pour l'opérateur de déplacement $SHC^{OOM}(V^*, i)$. Ce choix est effectué par tirage aléatoire uniforme sans remise. Le chemin parcouru en utilisant cet opérateur dépend de l'ordre dans lequel les v_i sont choisis. Cette observation signifie que le point de convergence d'une même graine ne sera pas forcément le même pour différentes optimisations indépendantes.
Dans l'utilisation courante, ce grimpeur est utilisé à partir d'un ensemble de graines et la meilleure solution est mémorisée. La stratégie consistant à choisir des graines aléatoirement dans l'ensemble $\mathcal{P}$ est appelée la "stratégie P".

Comparaison avec le Gibbs Site Sampler

Le grimpeur que nous avons décrit est très similaire au procédé qu'utilise le GSS (Lawrence et al., 1993). Ce programme, bien que toujours présenté sous un point de vue statistique, correspond à un grimpeur stochastique sur les voisinages $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$. Les différences sont que le GSS ne fonctionne que sous le mode OOPS, ou sous des contraintes que l'utilisateur doit spécifier précisément (nombre d'occurrences exactes dans chaque séquence).

D'autres différences concernent le procédé d'optimisation lui-même. L'évaluation du voisinage $\mathcal{N}^{OOM}$ par exemple n'est pas effectué directement sur la fonction objectif mais sur le ratio des vraisemblances que le voisin (l'occurrence potentielle) ait été généré par le modèle estimé sur toutes les autres occurrences (le procédé est expliqué plus en détail dans

la Section 3.5.1). Le choix du déplacement est effectué aléatoirement par une sélection de type "roue de la loterie". Il apparaît que l'évaluation sur le ratio de vraisemblances n'est pas strictement équivalente au score objectif. Dans le cas d'un signal fort, cela ne fait probablement pas ou très peu de différences, mais si le signal est faible, les choix de déplacement seront probablement très différents.
Le même procédé aléatoire est utilisé pour l'opérateur de décalage "phase shift". Un alignement n'est donc pas systématiquement "recalé".

4.6.3 Deux stratégies supplémentaires pour le mode OOPS

Introduction

Nous avons présenté dans la Section 4.6.2 la stratégie P qui consiste à lancer un grimpeur sur des graines aléatoires issues de l'ensemble $\mathcal{P}$. Bien que simple et directe, elle donne de très bons résultats dans la pratique. Les deux stratégies supplémentaires que nous présentons dans cette Section ne concernent que le mode OOPS, et traitent de la manière de choisir les graines à optimiser. Nous avons ainsi cherché à effectuer ces choix d'une façon plus pertinente qu'un simple tirage aléatoire des graines.

La deuxième stratégie consiste à construire des graines à partir de mots aléatoires. Ces mots, de longueur W, et qui peuvent n'avoir aucune occurrence exacte dans S, sont considérés comme des ancêtres hypothétiques. Une graine est construite à partir du mot choisi en assemblant les facteurs de S^* considérés comme ses "descendants" potentiels. Des ULMAs (graines) sont construits en assemblant des facteurs s^* qui présentent une similarité avec le mot ancêtre. Ils présentent de ce fait une valeur objectif attendue considérablement plus élevée qu'un ULMA quelconque choisi au hasard. Ces graines ainsi construites sont utilisées comme "graines prometteuses" par le grimpeur.
Nous présentons également une troisième stratégie qui met en oeuvre un algorithme évolutionniste sur l'espace des mots de longueur W, afin de choisir des mots pertinents et améliorer de ce fait la qualité des graines produites.
Nous proposerons dans la Section 5.3 une comparaison de ces trois stratégies selon le point de vue du temps CPU ainsi que du nombre d'évaluations requis de la fonction objectif pour atteindre un résultat donné. Nous discuterons également d'un problème "challenge" lancé à la conférence ISMB 2000 (Pevzner and Sze, 2000), et montrerons que l'utilisation de l'algorithme évolutionniste permet de résoudre efficacement certaines des instances de ce problème. Nous montrerons également qu'une grande partie des publications qui sont dédiées au "challenge" présentent des résultats fortement biaisés en faveur des algorithmes dédiés.

La stratégie utilisant l'algorithme évolutionniste a été implémentée sous le nom "MODEL" (Motif Discovering with Evolutionary Learning) (Hernandez et al., 2002; Hernandez et al., 2004), et est disponible sur requête. Une interface WEB a également été implémentée et est disponible à l'adresse suivante :
http ://idefix.univ-rennes1.fr :8080/PatternDiscovery/. Nous remercions à ce propos, Emmanuelle Morin, qui a réalisé cette implémentation dans le cadre du groupe "Symbiose" (IRISA, Rennes) dirigé par Jacques Nicolas. Les deux stratégies qui vont être présentées ont été développées en utilisant uniquement l'entropie relative $OF^{(H)}$ comme fonction objectif. La raison en est que ces travaux sont postérieurs au développement de la nouvelle fonction $OF^{(HR)}$. Il n'y a cependant pas de contraintes particulières qui empêcheraient

l'utilisation de l'entropie relative recouvrante pour ces deux stratégies supplémentaires.

La stratégie Q : projection de mots vers l'espace des alignements

La deuxième stratégie que nous proposons, appelée "stratégie Q", consiste à réduire l'espace des graines possibles à un sous ensemble $\mathcal{Q} \subset \mathcal{P}$, dont l'espérance des valeurs objectif de ses éléments est sensiblement plus élevée que pour ceux de $\mathcal{P}$. Le grimpeur n'est alors lancé que sur des graines choisies aléatoirement dans cet ensemble. L'ensemble $\mathcal{Q}$ est implicitement défini par un opérateur appelé $MtoP$ dit de "projection", qui permet de générer des éléments de cet espace. Soit $\mathcal{M} = \{M_1, M_2, \ldots, M_{(K^W)}\}$, l'ensemble de tous les mots de longueur W que l'on peut constituer sur un alphabet de taille K. L'opérateur $MtoP$ est une fonction injective $MtoP : \mathcal{M} \longrightarrow \mathcal{P}$ qui permet de produire un ULMA à partir d'un mot quelconque de longueur W. L'ensemble $\mathcal{Q}$ est défini par :

$$\mathcal{Q} = \{V \in \mathcal{P} \mid \exists M \in \mathcal{M}, MtoP(M) = V\}$$

$\mathcal{Q}$ correspond donc à l'ensemble des éléments de $\mathcal{P}$ qui sont atteignables par une projection à partir de n'importe quel élément de l'ensemble $\mathcal{M}$.
Nous définissons l'opération $MtoP(M) = V$ comme un procédé consistant à construire un ULMA en recherchant dans S^* les N facteurs qui sont maximalement similaires à M. L'idée est de considérer M comme un mot consensus et de rassembler certaines de ses occurrences dans S^* en un ULMA. Soit $Sc(M, s^*)$ un score de similarité entre M et s^*, défini par : $Sc(M, s^*) = W - H(M, s^*)$ avec $H(M, s^*)$ étant la distance de Hamming entre les deux mots (d'autres mesures de similarité pourraient également être utilisées). Soit D_i le domaine de v_i défini sous le mode OOPS par :

$$D_i = \{(i-1)(L-W+1)+1, \ldots, i(L-W+1)\}$$

L'opérateur $MtoP$ utilise M pour instancier chaque v_i de façon à ce que :

$$v_i = k \in D_i \mid \forall j \in D_i,\ Sc(M, s_k^*) \geq Sc(M, s_j^*)$$

L'ULMA V ainsi construit rassemblera pour chaque séquence le facteur qui maximise la similarité avec M. Cependant, étant donné que Sc ne peut prendre que $W+1$ valeurs différentes $\{0, 1, \ldots, W\}$ il sera fréquent que plusieurs facteurs issus d'une même séquence maximisent le score de similarité avec M, résultant en des ambiguïtés sur l'instanciation de certaines variables. Nous levons ces ambiguïtés à l'aide d'un algorithme glouton : l'ULMA partiel formé des v_i ne présentant pas d'ambiguïté sur leurs instanciations est construit. A ce stade, nous disposons d'un ULMA partiel dont le vecteur V n'est pas complètement instancié. Les variables restantes sont instanciées successivement en choisissant parmi l'ensemble des instanciations équivalentes du point de vue de Sc, celle qui maximise la fonction objectif de l'ULMA partiel. Bien que non-exacte, cette procédure s'est avérée suffisamment efficace. Il faut garder à l'esprit que le but de cet opérateur n'est pas d'optimiser un alignement, mais de produire des graines prometteuses pour le grimpeur. Le fonctionnement est décrit dans le Pseudocode 3. La Figure 4.8 propose une représentation graphique de cet opérateur.
Cet opérateur a été implémenté avec une complexité temporelle de $\mathcal{O}(WLT)$. Les distances de Hamming entre un mot de longueur W et les $(L-W+1)T$ facteurs de S^* doivent être calculées. Cette complexité pourrait cependant certainement être diminuée ou bornée en exploitant les répétitions dans les séquences.

Pseudocode 3 Opérateur $MtoP$: algoritme glouton

$\mathcal{U}$: Un ensemble d'occurrences formant un ULMA
$\mathcal{A}_i$: L'ensemble des instanciations équivalentes pour v_i
$\mathcal{A}_{i,k}$: Le k^{ieme} élément de $\mathcal{A}_i$
Début
$\mathcal{U} = \emptyset$
POUR TOUS $i \in \{1..N\}$
  SI $|\mathcal{A}_i| = 1$
    $\mathcal{U} = \mathcal{U} \cup s^*_{\mathcal{A}_{i,1}}$
FIN POUR TOUS
POUR TOUS $i \in \{1..N\}$
  SI $|\mathcal{A}_i| > 1$
    $max = 0$
    POUR TOUS $k \in \{1..|\mathcal{A}_i|\}$
      $score = OF(\mathcal{U} \cup s^*_{\mathcal{A}_{i,k}})$
      SI $score > max$
        $max = score$
        $v = \mathcal{A}_{i,k}$
      $\mathcal{U} = \mathcal{U} \cup s^*_v$
    FIN POUR TOUS
FIN POUR TOUS
fin

La stratégie Q consiste alors à produire des graines à partir de mots aléatoires en utilisant l'opérateur $MtoP$. Ces graines sont successivement optimisées par le grimpeur décrit précédemment.

La stratégie AG : un algorithme évolutionniste sur les mots

La troisième stratégie que nous proposons, appelée "stratégie AG", met en oeuvre un algorithme évolutionniste sur l'ensemble $\mathcal{M}$. L'amélioration par rapport à la stratégie Q consiste à explorer $\mathcal{M}$ à l'aide d'opérateurs sur les mots, dans le but de converger vers des mots consensus pertinents par rapport à l'ensemble de séquences S, contrairement à la stratégie Q où ces mots sont choisis aléatoirement. Le principe est celui d'un algorithme génétique classique, en utilisant en alternance une phase de sélection des solutions potentielles (mots) et une phase de transformation de celles-ci. Une solution est un vecteur de W variables, lesquelles sont définies sur un domaine de taille K, correspondant au K symboles de l'alphabet utilisé. Nous utiliserons cependant, par souci de simplicité, la terminologie de "mot" et de "symboles" pour parler du vecteur et de l'état de ses variables. Un échantillon de mots (la population) est initialisé aléatoirement, puis à l'aide d'un opérateur de sélection et d'opérateurs de transformation (opérateurs génétiques), la population de taille tp est itérativement modifiée afin de converger vers un ensemble de mots "ancêtres" pertinents par rapport à l'ensemble de séquences S.

La fitness d'un mot M est donnée par $OF^{(H)}(MtoP(M))$. Cette valeur est interprétée comme le potentiel du mot à générer une bonne graine pour le grimpeur. Nous utilisons un opérateur de sélection par tournoi de taille 2 : deux mots sont aléatoirement choisis

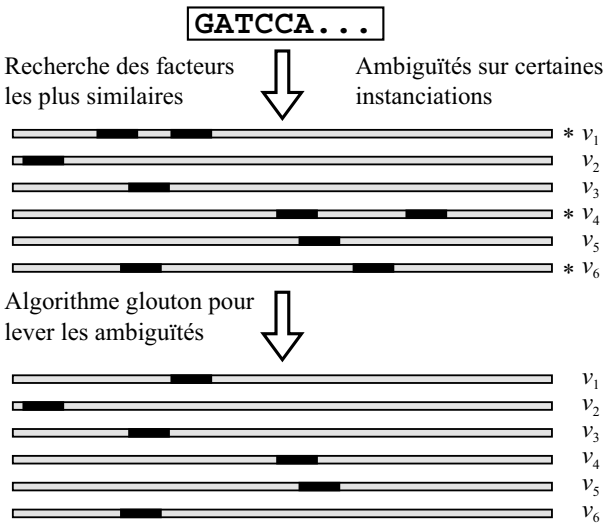


Fig. 4.8 : L'opérateur $MtoP$ permet de produire un ULMA à partir d'un mot M. Le mot est considéré comme un ancêtre hypothétique et l'opération consiste à rassembler dans un ULMA les facteurs qui maximisent un score de similarité avec M. Ces facteurs peuvent être vus comme des descendants de M.

dans la population et le meilleur des deux est retenu. Ce type de sélection a la propriété de ne dépendre que du rang des mots triés sur leurs valeurs de fitness. Cette propriété est souhaitable dans notre cas, car les valeurs de fitness peuvent présenter des écarts importants, provoquant une convergence prématurée vers un très petit nombre de solutions (Blickle and Thiele, 1995).

Nous utilisons six opérateurs sur les mots (Figure 4.10) dont trois sont tout à fait classiques des algorithmes génétiques : La recombinaison à un point (R1P), la recombinaison uniforme (RU) et l'opérateur de mutation (OM). Les trois opérateurs restants sont spécifiques au problème et ont de ce fait un rôle important dans l'efficacité de l'algorithme. Ces opérateurs sont le *glissement*, la *recombinaison homologue*, et le *réajustement* :

- *Opérateur de glissement* (OG). Cet opérateur consiste à décaler d'une position tous les symboles d'un mot, soit vers la droite, soit vers la gauche, avec la même probabilité (0.5). Le symbole éjecté est remplacé par un symbole aléatoire. Cet opérateur est tout comparable à l'opérateur de déplacement SHC^{PS} (phase shift) utilisé par le grimpeur. Il donne une chance à un mot de corriger un éventuel "mauvais calage".

- *La recombinaison "homologue"* (RH). Cet opérateur de recombinaison cherche le meilleur appariement possible entre les deux mots avant de procéder aux échanges de symboles. Soient $x = \{x_1, x_2, \ldots, x_W\}$ et $y = \{y_1, y_2, \ldots, y_W\}$ deux mots de longueur W. Un opérateur de recombinaison classique apparie les mots de façon à ce que x_1 soit en face de y_1, et ne procédera donc qu'à des échanges entre des symboles ayant la même position dans le mot. Définissons la notation $x_i \bowtie y_i$, pour décrire l'appariement des deux mots consistant à disposer x_i face à y_i. Les $2W-1$ appariements possibles correspondent à : $x_1 \bowtie y_W, x_2 \bowtie y_W, \ldots, x_{W-1} \bowtie y_W$ et $x_W \bowtie y_1, x_W \bowtie y_2, \ldots, x_W \bowtie y_W$. La recombinaison homologue évalue ces $2W-1$

Fig. 4.9 : L'opérateur $PtoM$ produit à partir d'un alignement V le mot le plus probable en fonction de la matrice des fréquences de V. Ce mot correspond donc à un consensus de l'alignement.

appariements et choisit celui qui possède le plus grand nombre de symboles identiquement appariés. Les échanges sont ensuite effectués avec une probabilité uniforme sur toute la partie recouvrante de l'appariement. Cet opérateur à été défini en partant de l'observation qu'une information "non homologue" risquait d'être échangée par les opérateurs de recombinaison classiques. Dans la problématique qui nous intéresse, un mot représente un ancêtre hypothétique qui possède des descendants sous la forme d'occurrences approchées dans l'ensemble de séquences (opérateur $MtoP$). Considérons deux mots qui ont les mêmes occurrences dans S à l'exception d'un décalage de quelques positions vers la droite ou la gauche. Ces deux mots contiennent une information similaire que nous qualifions *d'information homologue*, car elle porte sur les mêmes symboles de S. Une recombinaison classique va détruire cette information car elle présente un décalage entre les deux solutions. La recombinaison homologue cherche à identifier une éventuelle homologie avant d'effectuer les échanges.

- *L'opérateur de réajustement* (OR). Cet opérateur transforme un mot en utilisant l'espace des alignements $\mathcal{P}$. Soit un deuxième opérateur de projection $PtoM$, lequel est une fonction injective $PtoM : \mathcal{P} \longrightarrow \mathcal{M}$ qui à l'inverse de $MtoP$, produit un mot à partir d'un ULMA. Son fonctionnement consiste simplement à produire le mot consensus de l'ULMA. Ce consensus correspond au mot le plus probable en fonction de la matrice des fréquences observées F de l'ULMA. Le fonctionnement de cet opérateur est illustré dans la Figure 4.9. L'opérateur de réajustement d'un mot utilise d'abord $MtoP$ pour construire l'ULMA correspondant. Cet ULMA est ensuite optimisé par le grimpeur (voisinage $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$). L'opérateur $PtoM$ produira finalement le mot transformé à partir de l'ULMA convergé. Cet opérateur provoque souvent des changements importants sur le mot du point de vue de la distance de Hamming. Cependant, le mot obtenu est pertinent car il représente le consensus d'un alignement optimisé.

Étant donné que la fitness d'un mot correspond à la valeur objectif de l'ULMA correspondant (obtenu par l'opérateur MtoP), chaque mot a un ULMA qui lui est associé. A la fin de chaque génération, un nombre donné de ces ULMAs sont choisis comme graine pour le grimpeur strict. Le nombre de ces graines n'est pas explicitement fixé mais le grimpeur est appliqué de façon à ce que le temps CPU qui lui soit alloué corresponde aux deux tiers du temps CPU total (algorithme évolutionniste compris). Cet ajustement est empirique.

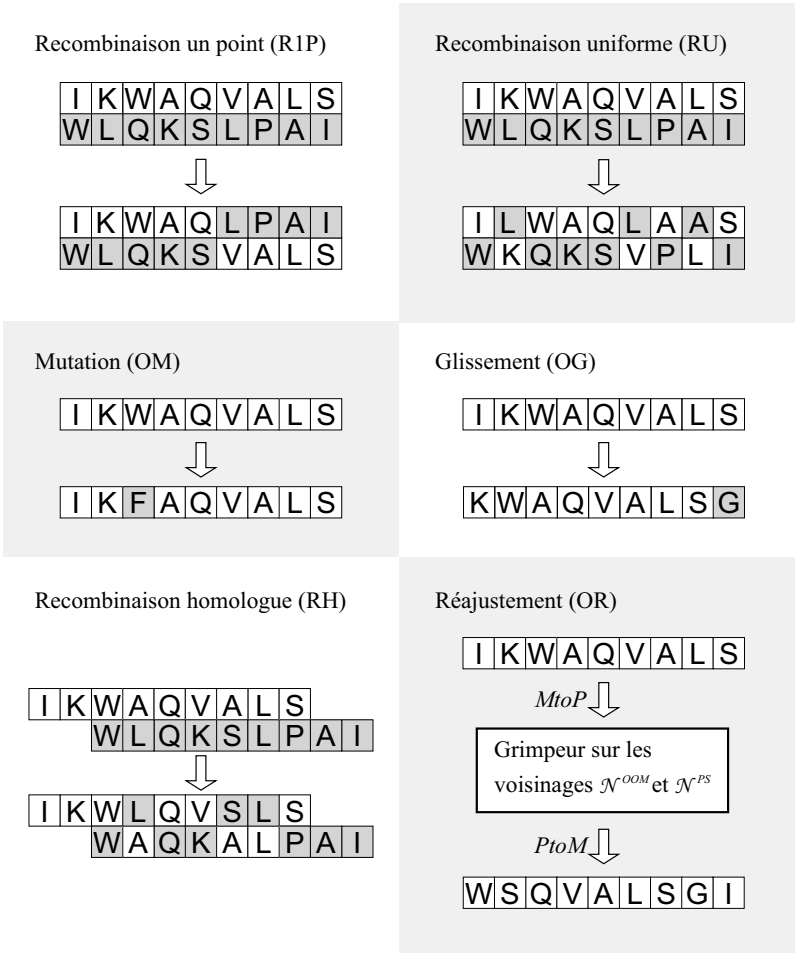


Fig. 4.10 : Les six opérateurs génétiques sur les mots sont présentés (les exemples de mots sont fictifs). En plus des opérateurs génétiques classiques, notons la recombinaison homologues qui apparie les deux mots de façon à faire correspondre l'information dite homologue avant l'échange. L'opérateur de réajustement provoque des changements drastiques du point de vue de la distance de Hamming. Le mot produit est cependant plus pertinent car il représente un consensus de l'alignement obtenu par le grimpeur.

Il arrive fréquemment qu'un mot soit rencontré un grand nombre de fois au cours d'une même optimisation par l'algorithme évolutionniste. Cette observation n'est pas gênante en soi : une même graine produite par un mot est optimisée par différents chemins lorsque plusieurs grimpeurs sont lancés sur cette graine, et produit au final des ULMAs qui ne sont pas forcément identiques. Cependant, afin d'éviter une ré-évaluation d'un mot rencontré, laquelle est coûteuse en temps CPU, tous les mots évalués sont mémorisés en leur associant leur valeur de fitness ainsi que l'ULMA leur correspondant. Nous utilisons pour cela une structure d'arbre. Cet arbre indexe les mots rencontrés en les représentant par un chemin partant de la racine jusqu'à une feuille. Chaque feuille représente donc un mot unique, et nous y associons les informations relatives (fitness et ULMA correspondant). Cette structure nous permet donc de vérifier rapidement si un mot a déjà été rencontré, et de récupérer les informations associées le cas échéant. Si le mot n'a jamais été rencontré auparavant, il est rajouté dans l'arbre avec ses informations associées. Citons notamment que cette structure à été utilisée dans le cadre d'un autre travail pour compter le nombre de fois qu'une solution a été rencontrée, permettant ainsi d'obtenir des informations sur la redondance d'une exploration (Martin et al., 2003; Gras et al., 2004). L'utilisation de cette structure nous permet en fonction de la redondance des mots explorés, un gain de temps CPU significatif, lequel est facilement de l'ordre de 20%. Par contre, les besoins en mémoire peuvent devenir importants dans le cas d'optimisations longues car tous les nouveaux mots sont mémorisés.

Le paramétrage de l'algorithme évolutionniste est empirique. Il est important de garder à l'esprit que son but n'est pas de converger vers le mot de valeur la plus élevée, mais de produire un ensemble de mots prometteurs. Le paramétrage de l'algorithme a donc été conçu dans cette optique de façon à ce que la population soit suffisamment modifiée à chaque itération. Nous n'avons en effet aucune garantie que le mot de fitness maximale produise la meilleur graine pour le grimpeur. Il pourrait même être possible que le mot maximisant la fitness ne fasse pas partie du bassin d'attraction de l'ULMA maximisant $OF^{(H)}$.
Les probabilités d'application des opérateurs ont été fixées de la manière suivante : 0.12 pour R1P, 0.1 pour RU (les symboles sont échangés indépendamment avec une probabilité de 0.30), 0.22 pour RH (avec probabilité d'échange des symboles de 0.30), 0.15 pour OG, 0.03 pour OR. L'opérateur de mutation initialise aléatoirement chaque symbole avec une probabilité de 0.08. Les opérateurs sont appliqués dans l'ordre suivant : OM, R1P, RU, RH, OG, OR. La taille de la population est fixée à $tp = 200$.

4.6.4 Résumé des trois stratégies et discussion

Nous avons décrit trois stratégies d'optimisation pour le problème de l'alignement local multiple sans indel. Ces trois stratégies utilisent toutes un grimpeur strict fonctionnant en alternance sur deux fonctions de voisinage $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$. Ce grimpeur a été décrit dans la Section 4.6.2. Les trois stratégies sont caractérisées par la manière de choisir des graines pour le grimpeur. La stratégie P est la plus simple. Elle consiste à lancer le grimpeur sur des éléments aléatoires de $\mathcal{P}$, l'ensemble des solutions possibles. Les deux autres stratégies n'ont été développées que pour le mode OOPS. La stratégie Q restreint le nombre de graines possibles à un sous ensemble $\mathcal{Q} \subset \mathcal{P}$. L'ensemble $\mathcal{Q}$ est implicitement défini par un opérateur $MtoP$ qui permet de générer des éléments de cet ensemble à partir de mots quelconques de longueur W. L'ensemble des W^K mots possibles est noté $\mathcal{M}$. La stratégie

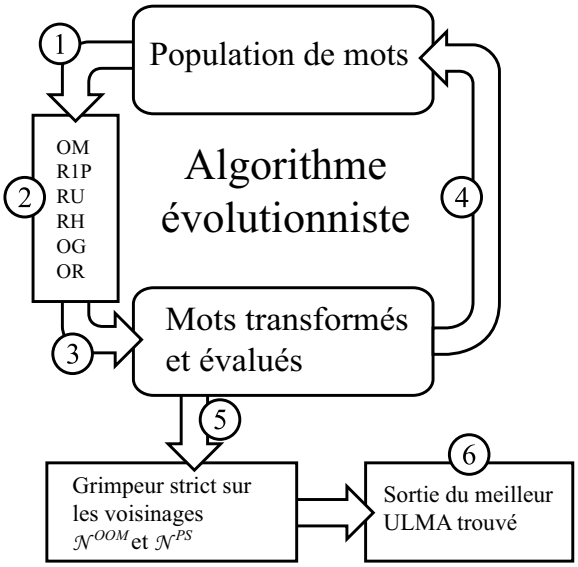


Fig. 4.11 : Schéma général de la stratégie AG utilisant un algorithme évolutionniste sur les mots. (1) Un opérateur de sélection par tournoi choisit avec remise tp mots dans la population. (2) Ces mots sont stochastiquement transformés par six opérateurs. (3) Les mots transformés sont évalués. Leurs valeurs de fitness correspondent à la valeur objectif de l'ULMA obtenu après l'application de l'opérateur $MtoP$. (4) La population est remplacée par les mots transformés. (5) Une sous-partie des ULMAs (graines) obtenus par la projection $MtoP$ est sélectionnée et est optimisée par le grimpeur. Le nombre de graines sélectionnées est calculé de façon à ce que le temps CPU qui soit alloué au grimpeur n'excède pas les deux tiers du temps CPU total. (6) Le meilleur ULMA est produit en sortie.

Q consiste à générer des graines à partir de mots aléatoires. La Stratégie AG utilise quant à elle un algorithme évolutionniste sur l'espace $\mathcal{M}$. Il cherche à converger vers des mots qui permettent de générer des graines prometteuses. Les deux dernières stratégies n'ont pas été développées pour les modes AMOOPS, ALOOPS et AOPS, car une adaptation efficace de l'opérateur $MtoP$ à ces modes n'est pas un problème trivial. Nous avons procédé à quelques essais concernant le mode AOPS. D'une façon générale, nous avons constaté que bien que non handicapantes, les deux stratégies Q et AG n'apportaient pas d'avantages significatifs par rapport à la stratégie P. Il faut savoir que les avantages qu'apportent ces deux dernières stratégies pour le mode OOPS, (résultats présentés dans la Section 5.3.3), reposent en grande partie sur la rapidité de cet opérateur et il nous parait difficile de l'adapter aux autres modes sans augmenter sa complexité temporelle.

Chapitre 5

Résulats

5.1 introduction

Cette Section propose différents résultats expérimentaux qui évaluent deux aspects dans la problématique de la découverte d'ULMA. Le premier aspect, et peut être le plus important concerne la qualité des fonctions objectif que l'on optimise. Nous avons mené différentes expérimentations comparatives entre l'entropie relative $OF^{(H)}$, classiquement utilisée par la communauté, et l'entropie relative recouvrante, fonction objectif que nous avons développée spécifiquement pour l'optimisation d'alignements sur des séquences protéiques. Nous les comparons sur leur capacité à isoler un signal du bruit, ainsi que sur leurs incidences sur le paysage d'exploration. Ces expérimentations sont effectuées sur des données réelles et artificielles. Nous montrons que notre fonction $OF^{(HR)}$ est préférable en terme de description du signal, ainsi qu'en terme de temps CPU requis pour une optimisation, et ce malgré sa complexité temporelle plus élevée. Le deuxième aspect traite des capacités exploratoires. Nous évaluons différentes stratégies, en incluant des méthodes classiquement utilisées par la communauté, sur leur capacité à converger vers un ULMA supposé de valeur objectif maximale.
Cette Section est subdivisée selon le type de données utilisées. La Section 5.2 traite d'un problème challenge qui a été proposé à la communauté travaillant sur la découverte de motifs. Ce challenge a suscité une littérature relativement importante et une grande partie des articles le concernant proposent des résultats biaisés en faveur d'algorithmes dédiés. Nous montrons que ce challenge n'est pas utilisable pour évaluer la plus grande partie des algorithmes non-dédiés. Une légère modification concernant la condition d'arrêt de la stratégie AG permet cependant de résoudre efficacement l'instance de base de ce challenge. La Section 5.3 présente des comparaisons sur des données réelles, impliquant un ensemble de protéines distantes connues pour contenir chacune une occurrence d'un motif hélice-tour-hélice (HTH). Nous y comparons sous le mode OOPS différentes méthodes classiques dédiées, ainsi que les deux fonctions objectif. La Section 5.4 propose des résultats de comparaison des deux fonctions, impliquant des données protéiques artificielles générées par Rose (Stoye et al., 1998), un simulateur d'évolution de séquences de biopolymères. Ce programme permet de générer à volonté des ensembles artificiels de protéines ayant une propriété d'homologie. Nous avons alors comparé les deux fonctions objectif, sur des données présentant des difficultés croissantes, impliquant une variation de la distance évolutive entre les séquences d'un ensemble. Nous montrons que la fonction $OF^{(HR)}$ est supérieure quant à sa capacité à reconnaître un signal dans un niveau de bruit élevé, tout en conférant au processus d'optimisation une performance plus élevée du point de vue du

temps CPU. La Section 5.5 concerne finalement un ensemble de protéines connues pour contenir chacune plusieurs occurrences d'un motif EF-hand. Nous illustrons les capacités d'optimisation sous les modes ALOOPS et AOPS et comparons plusieurs algorithmes classiques ainsi que les deux fonctions objectif.

5.2 Le problème challenge

Lors de la conférence ISMB 2000, un challenge concernant la réalisation d'un algorithme capable de retrouver un signal inséré dans un ensemble de séquences artificielles d'ADN a été lancé (Pevzner and Sze, 2000). Les données du challenge sont les suivantes :

1. Générer un ensemble de T séquences de longueur L sur un alphabet de cardinal $K = 4$ (ADN), tous les symboles étant équiprobables.
2. Générer sur le même alphabet un mot M de longueur W.
3. Créer T nouveaux mots $M_1, M_2, \ldots, M_T$, chacun étant une image imparfaite de M, avec d substitutions à des positions aléatoires.
4. Insérer chaque M_i dans la i^{eme} séquence à une position aléatoire (l'insertion se fait en remplaçant W symboles dans la séquence de façon à ce que la longueur finale reste L).
5. Étant donné les séquences, retrouver M, ce qui est équivalent à retrouver ses occurrences approchées $M_1, M_2, \ldots, M_T$ dans les séquences.

Le paramétrage de base du challenge est : $L = 600, T = 20, W = 15, d = 4$. On parle dans ce cas d'un (600,20,15,4)-signal. Un coefficient de performance PC pour évaluer la qualité d'un algorithme est donné par : $|Z_1 \cap Z_2| \,/\, |Z_1 \cup Z_2|$ avec Z_1 correspondant à l'ensemble des positions des occurrences insérées et Z_2 correspond à l'ensemble des positions découvertes par l'algorithme. Un certain nombre d'algorithmes spécifiques obtiennent de bonnes performances sur ce problème, en particulier PatternBranching (Price et al., 2003), Winnower (Pevzner and Sze, 2000), Multiprofiler (Keich and Pevzner, 2002), Projection (Buhler and Tompa, 2001; Buhler and Tompa, 2002) and SMILE (Marsan and Sagot, 2000; Marsan, 2002). Ce challenge a attiré notre attention car dans (Price et al., 2003; Pevzner and Sze, 2000; Keich and Pevzner, 2002; Buhler and Tompa, 2001), des comparaisons ont été effectuées entre des algorithmes dédiés au problème et des méthodes classiques qui optimisent un alignement sous la fonction d'entropie relative $OF^{(H)}$. De mauvaises performances sont reportées pour ces derniers sans autres explications. Il apparaît pourtant que les contraintes utilisées ne sont pas les mêmes pour les méthodes dédiées et les autres. Si la contrainte de répartition de une occurrence par séquence ainsi que de la longueur W du mot à retrouver sont utilisées par toutes les méthodes, les méthodes spécifiques au problème challenge bénéficient en plus du nombre d de substitutions que contient chaque occurrence du signal par rapport à M et utilisent un score directement dérivé de cette information. Nous avons ainsi facilement montré (Hernandez et al., 2004) que sans cette information, un (600,20,15,4)-signal (qui correspond à un paramétrage du challenge jugé facile) se trouve au niveau du bruit et ne peut donc plus être retrouvé par des méthodes optimisant $OF^{(H)}$. Nous avons ainsi constaté que ces algorithmes convergeaient en fait vers un alignement de valeur supérieure à celui composé des occurrences du signal $M_1, M_2, \ldots, M_T$. La Table 5.1 illustre clairement ce propos avec le GSS et MoDEL. Nous pouvons cependant observer que le coefficient de performance PC est d'environ 0.45, ce qui signifie qu'un ULMA optimisé avec l'entropie relative est en moyenne composé de 45% des instances du signal

inséré, les autres occurrences correspondant à du bruit. Cette observation nous a permis d'utiliser l'algorithme évolutionniste sur les mots (stratégie AG) pour retrouver un (600,20,15,4)-signal. Il nous a suffit pour cela de considérer tous les mots échantillonnés au cours de l'exécution de MoDEL. Pour chacun d'eux, nous vérifions s'il a une occurrence dans toutes les séquences avec exactement d erreurs, ce qui signifie que le signal a été retrouvé. Nous utilisons donc la contrainte d du nombre de substitutions uniquement dans la condition d'arrêt sans changer notre stratégie d'optimisation qui reste toujours aussi générique. De cette manière, MoDEL s'est avéré très efficace en étant capable de retrouver le signal en quelques secondes, c'est à dire avec une performance équivalente au meilleur algorithme spécifique publié à notre connaissance (Price et al., 2003). La majorité des autres algorithmes spécifiques requièrent un temps CPU de l'ordre de la minute ou plus, pour ce paramétrage. Le signal devient par contre rapidement imperceptible pour $OF^{(H)}$ pour des paramétrages plus difficiles, rendant ce challenge inadéquat pour les algorithmes génériques.

	$>$	$=$	$<$	PC	tCalc
MoDEL	0.99	0.01	0.00	0.46	27
GSS	0.99	0.00	0.01	0.44	31

Tab. 5.1 : Optimisation pour le problème challenge sous l'entropie relative $OF^{(H)}$. Les colonnes étiquetées '>', '=', '<', montrent respectivement la fréquence avec laquelle MoDEL et le Site Sampler convergent vers un ULMA de valeur plus élevée, égale, et plus petite que celui composé des occurrences du signal inséré. La colonne 'PC' donne le coefficient de performance et 'tCalc' donne le temps CPU moyen par optimisation en seconde. Les résultats présentés sont moyennés sur l'optimisation de 500 instances indépendantes du problème challenge sous les paramètres $L = 600, T = 20, W = 15, d = 4$. Dans quasiment toutes les instances de ce problème, il existe un ULMA de valeur objectif supérieure à celle de l'ULMA formé par le signal inséré. Il est clairement montré que pour les méthodes qui optimisent $OF^{(H)}$ (MoDEL, MEME, CONSENSUS) ou une fonction proche (GSS), un (600,20,15,4)-signal est au niveau du bruit.

5.3 Alignements de motifs HTH

Afin de comparer les capacités d'optimisation de différentes stratégies, nous avons constitué un ensemble de protéines distantes, chacune étant connue pour contenir une occurrence d'un motif hélice-tour-hélice (HTH). Ce motif est surtout présent dans des facteurs de transcription et leur confère la propriété de pouvoir effectuer une liaison spécifique à un double brin d'ADN. On le trouve cependant dans d'autres types de protéines comme les histones. Nous utilisons ces données pour comparer les deux fonctions objectif $OF^{(HR)}$ et $OF^{(H)}$, ainsi que pour évaluer les performances des différentes stratégies P, Q, AG avec les principaux algorithmes classiques dédiés à ce problème. Toutes ces comparaisons ne concernent que le mode OOPS.

5.3.1 Le motif HTH

Le motif HTH est trouvé dans une grande variété de protéines dont la fonction implique une liaison spécifique à l'ADN. On retrouve ce motif chez quasiment toutes les

espèces vivantes et approximativement 23 lignées différentes sont identifiées (Rosinski and Atchley, 1999). A l'intérieur de chaque lignée, les protéines sont très conservées, cependant, la divergence est presque maximale entre les protéines de lignées différentes. Le motif structural HTH consiste en deux hélices alpha séparées par une courte région "turn". Cette conformation permet à la protéine de lier un segment spécifique d'ADN. La spécificité de la liaison est assurée par la deuxième hélice alors que la première stabilise le complexe. Cette spécificité n'est quasiment jamais conservée entre les lignées. L'étude menée par Rosinski suggère que toutes les protéines HTH auraient évolué à partir d'un même ancêtre commun.

Dans le but de construire un ensemble de séquences sur lequel l'optimisation serait difficile (condition requise pour mettre en évidence les capacités d'optimisation), nous avons constitué cet ensemble avec 16 protéines, chacune appartenant à une lignée différente. Les protéines choisies sont celles citées en exemple dans (Rosinski and Atchley, 1999) pour illustrer les différentes lignées. Étant donné que les méthodes que nous évaluons recherchent des motifs sans indel, nous n'avons retenu que les seize protéines dont le motif HTH peut être aligné sans indel, selon l'alignement proposé dans l'étude de Rosinski. Nous obtenons de cette manière un ensemble de protéines peu conservées mais supposées homologues, chacune contenant une occurrence connue du motif HTH. L'ensemble de séquences obtenu a une longueur moyenne de 217 acides aminés ($min = 61$, $max = 613$). La longueur d'un motif HTH est d'environ 22 acides aminés, tel qu'indiqué dans les annotations de SWISS-PROT. L'alignement supposé maximum obtenu par une optimisation de la fonction $OF^{(HR)}$ et pour une largeur $W = 22$ est noté V_{HTH}. Cet ULMA présenté dans la figure 5.1, est conforme aux annotations de SWISS-PROT, et correspond à une exception près à l'alignement proposé dans l'étude de Rosinski. Cette exception concerne la protéine (H4_CAEEL,P02306), une histone du nématode *Caenorhabditis elegans*. L'occurrence que nous retrouvons n'est pas la même que celle proposée dans l'étude de Rosinski, et aucun motif HTH n'est annoté pour cette protéine dans SWISS-PROT. Nous avons cependant observé que l'occurrence de H4_CAEEL alignée dans V_{HTH} est beaucoup plus significative que celle proposée par l'étude. De plus une comparaison avec deux structures tridimensionnelles de protéines très similaires indiquent que le domaine retrouvé correspond à une structure de type hélice-tour-hélice. Cependant, notre but ici n'est pas de faire une étude évolutionnaire des HTH mais de mesurer des capacités d'optimisation et cette question reste donc ouverte.

Il est également utile de mentionner que nous avons observé que ces séquences sont trop dissimilaires pour être alignées par des méthode classiques utilisant des comparaisons deux à deux de séquences telles que ClustalW ou T-COFFEE. Les méthodes d'inférences de modèles déterministes (découverte de patterns) ont également échoué à localiser ce motif, car il est trop peu conservé pour pouvoir être modélisé par un consensus. Ainsi, l'alignement multiple proposé dans l'étude de Rosinski n'a pas été obtenu à partir des séquences brutes des protéines HTH. Les régions contenant les occurrences du motif ont été manuellement extraites sur la base de connaissances expertes. L'alignement proposé a été produit pas ClustalW à partir de ces sous séquences extraites, lesquelles ont une longueur de 30 acides aminés, et correspondent aux occurrences connues du motif HTH.

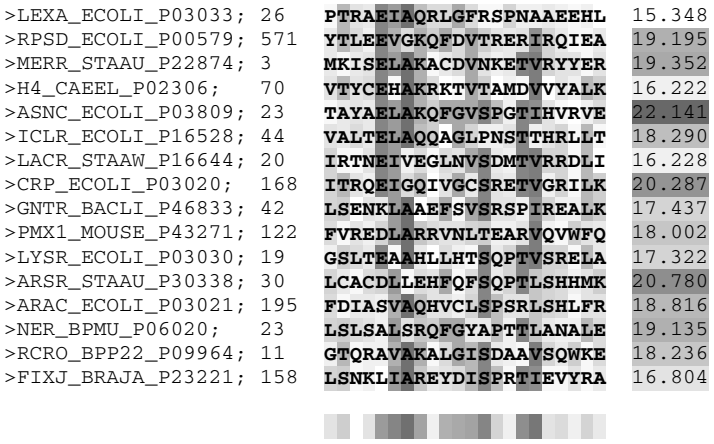


Fig. 5.1 : Alignement des domaines HTH obtenus par une optimisation de la fonction objectif $OF^{(HR)}$. Cet alignement a été produit sur un ensemble de 16 protéines correspondant chacune à une lignée distincte du motif HTH. Les entrées de chaque séquence dans la base de donnée SWISS-PROT sont indiquées à gauche. Viennent ensuite les positions de chaque occurrence dans les séquences, suivies de l'occurrence elle-même. Finalement est indiqué le logarithme du ratio des vraisemblances de chaque occurrence. Cet alignement noté V_{HTH} est supposé maximum pour $OF^{(HR)}$.

5.3.2 Comparaisons des fonctions sur les données HTH

Notre première comparaison sur les données HTH implique notre nouvelle fonction objectif $OF^{(HR)}$ que nous comparons avec $OF^{(H)}$, la fonction d'entropie classiquement utilisée pour ce type de problème. Nous montrons d'abord que notre fonction est beaucoup moins sensible au paramètre W qui spécifie la largeur de l'ULMA à optimiser. Nous comparons ensuite ces fonctions du point de vue de l'efficacité de l'optimisation et montrons que $OF^{(HR)}$ est largement avantageuse sur ces données.

Sensibilité au paramètre W

Un point sensible des méthodes d'optimisation d'ULMA concerne le paramètre W (largeur), que l'utilisateur doit spécifier. En effet, l'alignement présenté dans la Figure 5.1 a été obtenu par une optimisation de $OF^{(HR)}$ avec $W = 22$. Or, il apparaît que la valeur objectif $OF^{(H)}(V_{HTH})$ de cet ULMA correspond en fait à un point sous-optimal de l'espace de recherche. Il existe en effet un autre ULMA supposé optimal[1] pour $OF^{(H)}$ et $W = 22$, qui ne contient en fait que quatorze des seize occurrences du motif. Nous avons alors mené des optimisations des deux fonctions $OF^{(HR)}$ et $OF^{(H)}$ sous différentes valeurs de W sur les données HTH. Ces optimisations ont été conduites de façon intensive, de façon à pouvoir supposer que les ULMAs obtenus ont une valeur objectif maximale. Ces optimisations ont été menées pour les deux fonctions avec $W \in \{12..30\}$. Nous avons alors compté dans chacun des ULMAs maximums obtenus le nombre d'occurrences du motif HTH correctement alignées, sur les 16 occurrences connues. Le résultat est présenté sous

[1] L'optimalité d'un ULMA ne peut pas être démontrée, mais nous supposons une optimalité lorsque aucun ULMA de valeur supérieur n'a pu être identifié au cours de nos nombreuse expérimentation utilisant différentes méthodes d'optimisation.

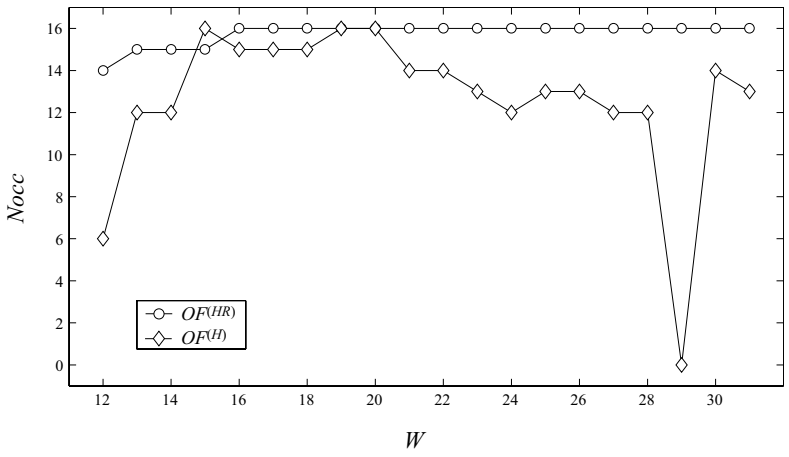


Fig. 5.2 : Ce graphique montre pour les deux fonctions $OF^{(HR)}$ et $OF^{(H)}$, le nombre d'occurrences du motif HTH qui sont correctement alignées pour différentes valeurs de W. On peut observer que W a une forte influence sur $OF^{(H)}$, qui préfère souvent attribuer une valeur objectif plus élevée à un ULMA qui n'est pas composé des seizes occurrences du motif HTH, conduisant ainsi l'optimisation vers un alignement en partie incorrect du point de vue biologique. La fonction $OF^{(HR)}$ est quant à elle très peu sensible à ce paramètre.

la forme d'un graphique dans la Figure 5.2. En abscisse sont indiquées les différentes valeurs de W, et en ordonnée le nombre d'occurrences correctement identifiées. On constate que $OF^{(HR)}$ est beaucoup moins sensible à ce paramètre que $OF^{(H)}$, en étant capable d'identifier les seize occurrences du motif pour quasiment toutes les valeurs W testées. Le nombre d'occurrences identifiées par $OF^{(H)}$ varie beaucoup et de façon irrégulière en fonction de W. Cette observation signifie qu'un utilisateur ayant peu de connaissances sur ses données risque fort de spécifier une largeur conduisant à un alignement incorrect avec $OF^{(H)}$.
Étant donné que nous allons utiliser par la suite ces données pour comparer plusieurs méthodes qui optimisent $OF^{(H)}$ ou une fonction très proche, nous avons choisi de considérer l'ULMA obtenu avec $W = 20$, comme l'ULMA de référence noté V_{ref}, lequel correspond à un alignement correct et supposé maximum pour les deux fonctions.

Comparaison d'optimisation avec les deux fonctions

Nous avons comparé l'influence des deux fonctions sur le processus d'optimisation par le grimpeur strict (stratégie P). Le procédé utilisé consiste à comparer les optimisations des deux fonctions sur leur taux de succès (retrouver V_{ref}) en fonction du temps CPU et du nombre d'évaluations de la fonction objectif. L'expérimentation est menée de la façon suivante : deux cent optimisations indépendantes sont effectuées pour chaque fonction, chacune pendant un temps CPU spécifié. Pour chaque optimisation, l'ULMA maximum atteint ainsi que le nombre d'évaluations déjà effectuées sont mémorisés à des intervalles de une seconde CPU. Ces données nous permettent ensuite de calculer un taux de succès en fonction du temps CPU et du nombre d'évaluations écoulées. Un succès signifie que V_{ref} a été retrouvé. Le taux est simplement calculé par le nombre de fois qu'un succès a été atteint à un temps ou à un nombre d'évaluations donné, divisé par le nombre total

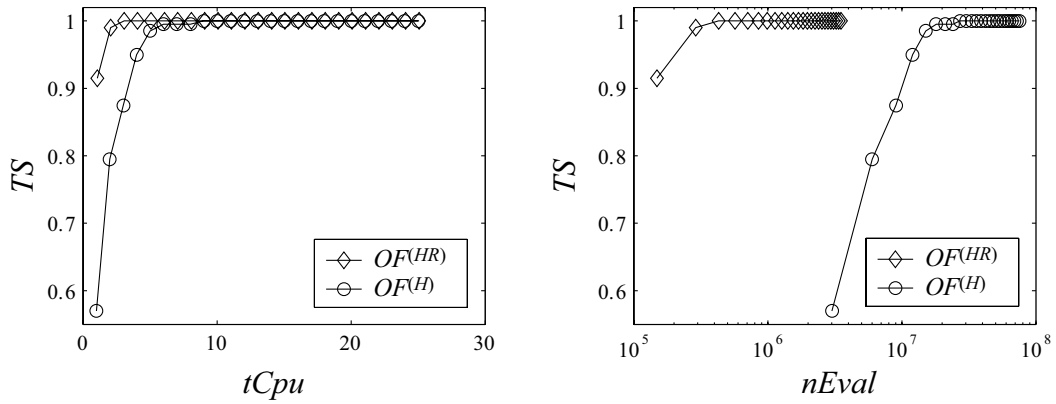


Fig. 5.3 : Ce graphique montre pour les deux fonctions $OF^{(HR)}$ et $OF^{(H)}$, le taux de succès (TS) exprimé en fonction du temps CPU ($tCPU$), et du nombre d'évaluations de la fonction objectif ($nEval$). Le nombre d'évaluations est donné sur une échelle logarithmique. On observe un net avantage pour l'entropie recouvrante.

d'optimisations indépendantes. Les résultats sont donnés sous la forme de graphiques dans la Figure 5.3. On observe que l'utilisation de l'entropie recouvrante est avantageuse sur les deux points de vues. Le nombre d'évaluations requis pour un taux de succès donné est d'environ soixante fois inférieur pour $OF^{(HR)}$. Cet avantage a une conséquence directe sur le temps CPU, permettant ainsi de retrouver V_{ref} environs trois fois plus rapidement, bien que son temps d'évaluation soit supérieur. On observe un net avantage pour l'entropie recouvrante.

5.3.3 Comparaisons d'optimisation avec les méthodes classiques

Retrouver l'ULMA référence

Notre première expérience consiste simplement à vérifier si les algorithmes classiques sont capables de retrouver l'alignement de référence V_{ref}. Pour cette comparaison, nous avons utilisé en plus de notre implémentation MoDEL, correspondant à la stratégie AG sous $OF^{(H)}$, le Gibbs Site Sampler (GSS) (Lawrence et al., 1993), MEME (Bailey and Elkan, 1995) et CONSENSUS (Hertz et al., 1990). Ces trois méthodes optimisent l'entropie relative ou une fonction proche (GSS). Nous nous sommes cependant assurés que ce programme ne converge pas vers un alignement différent présentant une valeur objectif plus élevée que celle de V_{ref}. Nous avons utilisé l'option de MEME qui le contraint à rechercher exactement une occurrence par séquence, les deux autres programmes le faisant par défaut. Le Gibbs Motif Sampler (GMS) n'a pas été pris en compte dans cette comparaison, car il ne peut pas être paramétré de façon à contraindre la répartition des occurrences à exactement une par séquence. Cette contrainte est indispensable sur ces données afin de pouvoir identifier les occurrences connues du motif. Sans l'information de la répartition des occurrences, le signal se retrouve au niveau du bruit, ne pouvant plus être identifié.

Dans le but d'augmenter les performances de ces algorithmes, nous avons intensivement cherché à optimiser leurs paramètres d'exploration. Pour le Gibbs Site Sampler, nous avons ajusté sa limite de convergence avec l'option '-L100' (optimise une nouvelle graine aléatoire si aucune amélioration ne s'est produite durant les cent dernières itérations).

Cette option a considérablement amélioré ses performances par rapport à celles obtenues avec la valeur proposée par défaut ('-L10'). Le programme CONSENSUS (en plus du paramétrage par défaut) a été lancé avec l'option '-q 30000'. Ce programme utilise un algorithme de recherche par faisceau et cette option spécifie la largeur du faisceau, soit le nombre de noeuds de l'arbre de recherche qui sont considérés à chaque profondeur. Cette valeur augmente linéairement le nombre de solutions partielles considérées, tout en augmentant la mémoire requise par l'algorithme. Nous avons cherché à augmenter cette valeur jusqu'à la limite mémoire de notre machine (2Go) sans pour autant améliorer le résultat obtenu. Concernant le programme MEME, la modification de ses paramètres n'a eu aucune influence sur le résultat produit, nous l'avons donc lancé avec les paramètres par défaut. Nous avons également mesuré une similarité entre l'ULMA de référence V_{ref} et celui trouvé par chaque méthode (V_{max}). Nous utilisons pour cela une mesure qui donne le taux de paires d'appariement identiques entre deux alignement (Sauder et al., 2000). Ce taux $sim(V_{ref}, V_{max})$, est calculé par le nombre d'appariements de symboles deux à deux, identiques entre V_{ref} et V^*, divisé par le nombre d'appariements totaux contenus dans l'alignement de référence, nombre qui dans notre cas est identique pour les deux alignements comparés soit $W((N(N-1))/2)$. Les résultats ont montré que malgré nos efforts d'ajustement des paramètres d'optimisation, MEME et CONSENSUS ne retrouvent pas V_{ref}, mais convergent toujours vers un sous-optimum. Ces résultats sont résumés dans la Table 5.2.

	$OF^{(H)}(V_{max})$	$sim(V_{ref}, V_{max})$	$tCpu$
MoDEL	32.71	1.00	4
Site Sampler*	32.71	1.00	13
CONSENSUS	31.96	0.01	5
CONSENSUS*	32.42	0.65	65
MEME	28.38	0.41	1

Tab. 5.2 : Comparaison de la capacité d'optimisation de plusieurs algorithmes sur les données de protéines contenant un domaine HTH. Les programmes impliqués sont MoDEL (stratégie AG sous $OF^{(H)}$), le Gibbs Site Sampler, CONSENSUS et MEME. L'étoile (*) indique les programmes qui ont été lancés avec un paramétrage différent que celui proposé par défaut. MEME n'est lancé qu'avec ses paramètres par défaut car nous n'avons pas pu améliorer ses performances en les modifiant. Le score maximum obtenu est indiqué dans la colonne "$OF^{(H)}(V_{max})$" et le temps CPU en secondes dans la colonne "$tCpu$" (CPU Athlon 1,6 GHz). La colonne "$sim(V_{ref}, V_{max})$" donne une mesure de similarité entre l'ULMA obtenu (V_{max}) et celui de référence (V_{ref}). Cette valeur correspond au taux de paires de symboles identiquement alignés entre les deux ULMAs. On constate que MEME et CONSENSUS n'arrivent pas à converger sur V_{ref}, seul MODEL et le Gibbs Site Sampler retrouvent l'alignement de référence.

Comparaisons d'optimisation des différentes stratégies

Nous avons effectué une comparaison plus complète impliquant le GSS et les trois stratégies P, Q, AG que nous proposons. Nous ne prenons pas en compte MEME et CONSENSUS dans cette comparaison car ils n'arrivent pas à retrouver V_{ref}. Ces quatre stratégies ont été comparées sur leurs taux de succès en fonction du temps CPU et du nombre d'évaluations requises, en suivant le même procédé que celui décrit précédemment

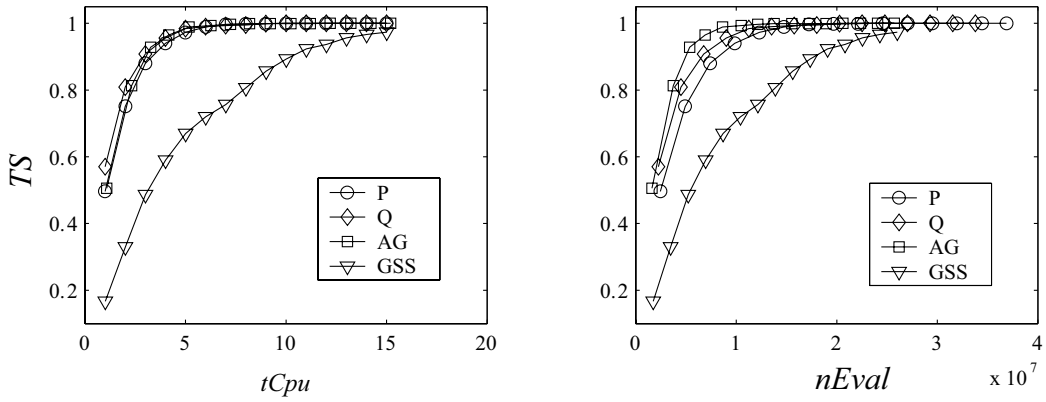


Fig. 5.4 : Comparaison de quatre stratégies d'optimisation P, Q, AG, et 'gibbs' pour le GSS, sur l'ensemble de protéines contenant un domaine HTH. Les deux graphiques représentent en ordonnée le taux de succès (retrouver V_{ref}) et respectivement en abscisse le temps CPU en secondes et le nombre d'évaluations de la fonction objectif.

(Section 5.3.2). Nous avons procédé à deux cents optimisations indépendantes de vingt secondes CPU sur les données HTH pour chacune des quatre stratégies. Un succès signifie que V_{ref} a été retrouvé. Les résultats sont présentés dans la Figure 5.4. Comme auparavant, le GSS a été paramétré avec l'option '-L100', qui s'est avérée être significativement meilleure que celle proposée par défaut ('-L10'). Nous pouvons observer que le GSS (principalement un grimpeur stochastique) a besoin d'évaluer approximativement trois fois plus de solutions pour un même taux de succès comparé au grimpeur strict. Les améliorations apportées par les stratégies Q et AG ne sont pas significatives au niveau du temps CPU. Par contre la stratégie AG évalue environ deux fois moins de solutions pour un même taux de succès comparé à la stratégie P.

5.4 Comparaison des fonctions $OF^{(HR)}$ et $OF^{(H)}$ sur données artificielles

Afin d'effectuer une comparaison plus large des deux fonctions $OF^{(H)}$ et $OF^{(HR)}$, nous avons utilisé des ensembles de séquences artificielles générés par le programme Rose (Stoye et al., 1998), qui simule un processus évolutionnaire sur des séquences nucléiques ou protéiques. L'approche consistant à utiliser des séquences artificielles a l'avantage de permettre de générer autant d'ensembles de séquences que nécessaire, ainsi que de faire varier la difficulté qu'ils peuvent présenter à un processus d'optimisation. Il faut cependant garder à l'esprit que bien que générés selon un modèle évolutionniste, ces ensembles de séquences restent une approximation simplifiée des caractéristiques présentes dans des données réelles.

5.4.1 Le générateur Rose

Rose permet de produire à partir d'une séquence ancestrale un ensemble de séquences "homologues" en simulant un processus évolutionnaire. Le programme propose un choix de paramètres importants. Nous ne décrirons cependant que les fonctionnalités que nous avons utilisées. Le programme commence par construire un arbre évolutionnaire uniforme

binaire de profondeur 9 soit $2^{9+1} - 1 = 1023$ noeuds. Une séquence ancestrale de longueur choisie L est associée à la racine de l'arbre. Cette séquence sera l'ancêtre commun de toutes les séquences qui seront générées. Cette séquence ancestrale est générée aléatoirement en utilisant une distribution de probabilité pour les acides aminés. La distribution utilisée est celle décrite dans (Dayhoff et al., 1978). Une distance évolutive est associée aux branches (la même distance pour toutes les branches). L'unité utilisée est le PAM ((Percent Accepted Mutation). Une distance de 1 PAM correspond à une mutation attendue par 100 acides-aminés. La distance associée aux branches est approximée à partir d'un paramètre appelé d_{av}, de façon à ce que la distance entre deux feuilles prises au hasard corresponde en moyenne à d_{av}. L'arbre ainsi construit comporte 512 feuilles. Afin d'obtenir le nombre de séquences T requis, T feuilles sont choisies aléatoirement et les autres sont supprimées. Les noeuds internes qui ne sont plus nécessaires (ne possèdent plus qu'un seul fils) sont également supprimés, mais en conservant les distances évolutionnaires initiales. Les noeuds inutiles disparaissent mais les longueurs des branches restent identiques. L'arbre ainsi obtenu servira de guide de mutation pour générer les séquences filles et représentera donc l'histoire de l'évolution de l'ensemble de séquences homologues qui sera généré. La Figure 5.5 présente un exemple d'arbre généré par Rose avec $T = 16$ et $d_{av} = 1000$.

Une fonction : $s_{fille} = evolue(s_{mere})$ est appliquée récursivement dans l'arbre afin de générer une séquence (fille) dans chaque noeud à partir de son ancêtre direct (mère). Cette fonction fait subir trois types de transformation à s_{mere} : les *substitutions*, les *insertions* et les *délétions*.

- *Les substitutions* : une fonction, $s_{fille}(i) = subst(s_{mere}(i), d)$ est appliquée sur toutes les positions i de s_{mere} afin de générer la séquence fille à une distance évolutive d souhaitée. Le programme utilise une matrice D, de taille 20X20, où chaque entrée $D(i, j)$ donne la probabilité que σ_j (le j^{eme} symbole de l'alphabet), soit substitué par σ_i, et pour une unité de distance (PAM 1). La somme des probabilités d'une colonne vaut 1.0 : $\forall j \in 1..K, \sum_{i=1}^{K} D(i, j) = 1.0$. Les probabilités $D(i, i)$ situées sur la diagonale représentent le degré de stabilité. Si par exemple, pour toutes les valeurs $D(i, i) = 0.99$, il en résultera que pour une unité de distance, seul un pourcent des symboles subiront une substitution. Le programme utilise par défaut la matrice de probabilité de distance PAM 1 décrite dans (Dayhoff et al., 1978), et répète la procédure autant de fois que nécessaire jusqu'à atteindre la distance d spécifiée par la longueur de la branche concernée dans l'arbre. Le fait d'utiliser des probabilités de substitution biaisées simule implicitement le phénomène de sélection naturelle. En effet, dans la nature, les probabilités de mutations ne sont pas biaisées en fonction des symboles, mais sont plus ou moins aléatoires dans l'ADN. Une mutation impliquant la substitution d'un acide aminé et altérant de ce fait gravement la fonction d'une protéine sera éliminée d'elle même car l'organisme concerné ne pourra pas survivre et la transmettre à d'éventuels descendants. Le programme ne peut évidement pas simuler cette sélection explicitement, mais il la modélise directement par les biais décrits dans la matrice de substitution utilisée.
- *Les insertions et les délétions* : le processus de création d'indels est plus empirique. L'utilisateur spécifie les probabilités p_{ins} (insertion) et p_{del} (délétion), avec laquelle chaque position de s_{mere} subira une insertion ou une délétion pour une distance PAM 1. L'utilisateur spécifie également une distribution de longueurs attendues pour les insertions et les délétions.

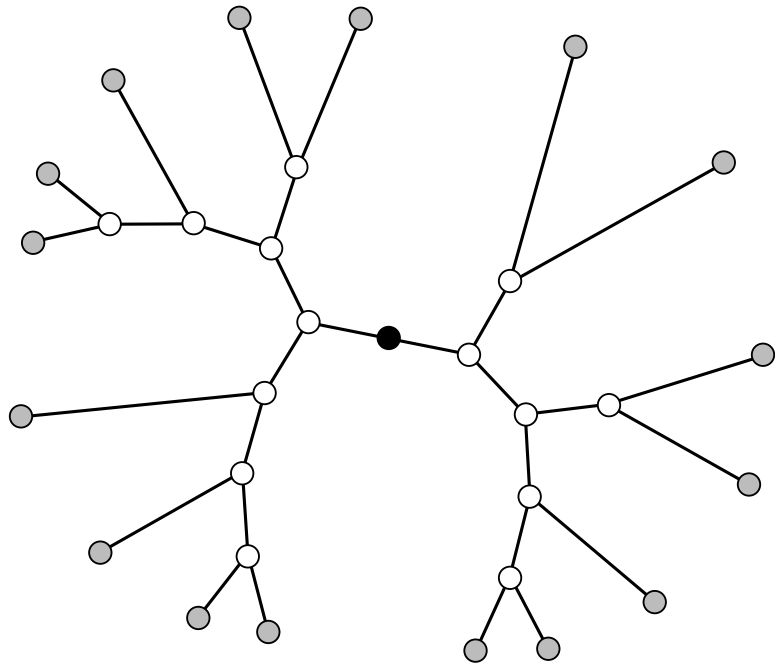
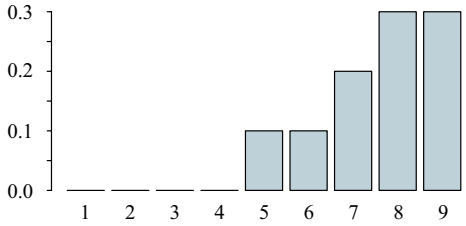


Fig. 5.5 : Un exemple d'arbre évolutionnaire généré par le programme Rose avec les paramètres $T = 16$ et $d_{av} = 1000$. Chaque noeud de l'arbre représente une séquence et les branches entre les noeuds ont une longueur proportionelle à la distance évolutive qu'ils représentent. Les plus petites branches représentent une distance de 125 PAM. La longueur de toutes les branches est un multiple entier de 125 PAM. Ces distances sont déterminées par le programme de façon à approximer une distance moyenne d_{av} entre les feuilles prises deux à deux. La racine de l'arbre est spécifiée par le rond noir. Elle contient la séquence ancestrale commune. Une simulation d'évolution à partir de cette séquence ancestrale permet de générer récursivement des séquences filles dans les noeuds internes, jusqu'aux feuilles. Celles-ci contiendront les séquences de l'ensemble homologue généré. Les séquences prises deux à deux dans cet ensemble présenteront des degrés de divergence variables selon la distance à laquelle se trouve leur ancêtre commun le plus proche, mais elles seront toutes à la même distance de la séquence ancestrale, située à la racine de l'arbre.

Le programme propose également la possibilité de modifier plus ou moins la pression évolutive sur chaque site de la séquence ancestrale. Il faut pour cela spécifier une valeur réelle te_i qui permet d'augmenter ou de diminuer linéairement le taux d'évolution à une position donnée $s_{mere}(i)$ de la séquence mère. Une valeur inférieure à 1.0 diminue la distance PAM et supprime les mutations de type indels pour le symbole spécifié. On peut de cette manière en spécifiant des $te_i < 1.0$ pour $i \in \{1..W\}$, forcer le processus évolutif à conserver un motif de largeur W sans indels, dont les occurrences prises deux à deux auront une distance évolutive moyenne plus petite que pour le reste des séquences. Ces valeurs te_i doivent être spécifiées pour toutes les positions de la séquence ancestrale. Elles sont ensuite reproduites pour les séquences filles. Dans le cas d'une délétion, les te_i correspondants sont supprimés, et des $te_i = 1$ sont ajoutés aux positions correspondant à une insertion.

5.4.2 Méthode

Nous avons généré huit séries d'ensembles de séquences : $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_8$. Chaque série est composée d'un nombre donné d'ensembles de séquences : $\mathcal{S}_i = \{S_{i,1}, S_{i,2}, \ldots, S_{i,Z_i}\}$, et tous les ensembles de séquences d'une série donnée sont générés sur un même paramétrage de Rose. Chaque série est générée en utilisant son propre paramètre d_{av}, de distance moyenne attendue entre deux feuilles de l'arbre. Les valeurs d_{av} commencent à $d_{av} = 200$ pour $\mathcal{S}_1$ et sont incrémentées de 200 à chaque série pour finir à $d_{av} = 1600$ pour $\mathcal{S}_8$. Tous les autres paramètres utilisés sont identiques pour toutes les séries. Le nombre de séquences de chaque ensemble est fixé à $T = 16$ et leurs longueurs moyenne à $L = 400$. Les probabilités d'indels sont de : $p_{ins} = 0.01$ et $p_{del} = 0.005$. La distribution des longueurs l d'indels est identique pour les insertions et les délétions, et est donnée dans l'histogramme suivant :



Ainsi la longueur minimum d'un indel est de $l = 5$ et la longueur maximum de $l = 9$ avec une préférence pour les indels longs. Nous avons également associé à W positions contiguës de la séquence ancestrale une valeurs $te_i = 0.2$ afin de diminuer le taux d'évolution sur ces positions. Nous forçons de cette manière le processus évolutif à conserver un motif présentant un niveau de conservation plus important que le reste des positions, et ne présentant pas d'indels. Ce motif correspond donc dans l'ensemble de séquences à un ULMA de largeur 15 sous le mode OOPS. Cet ULMA, noté V_{ref}, correspond au *signal* qu'il faut retrouver. Nous avons ainsi généré 1000 ensembles de séquences pour les six premières séries $\mathcal{S}_1..\mathcal{S}_6$ et 2000 ensembles pour $\mathcal{S}_7$ et $\mathcal{S}_8$. Ces deux dernières comportent plus d'ensembles car il est plus difficile d'obtenir pour des valeurs d_{av} élevées des ensembles comportant un signal identifiable par rapport au bruit.
Nous simulons de cette manière l'évolution d'une famille de protéines qui contiennent chacune une occurrence d'un motif sur lequel la pression sélective est forte. Le fait de faire varier d_{av} pour les différentes séries nous permet de faire reculer dans le temps l'ancêtre commun des séquences homologues, qui deviennent ainsi de plus en plus divergentes.

Ce paramétrage est conçu pour générer des ensembles de séquences présentant des difficultés d'optimisation croissantes, contenant un signal de plus en plus faible. Il faut cependant savoir qu'il n'est pas possible de se faire une idée précise des propriétés des paysages produits suite à une optimisation sur les voisinages $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$ (nombre de maximum locaux, taille des bassins d'attraction). Le paramétrage que nous avons utilisé est empirique et basé sur des suppositions quand à son effet sur le paysage résultant. Le dosage du ratio W/L (largeur de V_{ref} sur la longueur moyenne des séquences) permet de contrôler le nombre de similarités aléatoires dans les séquences. Les probabilités d'indels p_{ins} et p_{del}, qui sont relativement élevées contribuent à renforcer le signal du motif conservé. Les insertions, lorsqu'elles sont effectuées sur un ancêtre proche, contribuent à insérer dans quelques séquences un mot similaire, résultant en une conservation locale qui peut correspondre à un maximum local dans les paysages d'explorations.

Des optimisations ont été lancées sur chacun des dix mille ensembles de séquences produits par Rose et pour les deux fonctions objectif $OF^{(H)}$ et $OF^{(HR)}$. Une optimisation consiste à lancer le grimpeur strict décrit dans la Section 4.6.2, successivement à partir de graines aléatoires (stratégie P), pendant un temps CPU minimum de 30 secondes (Athlon 1.6 Ghz). Nous appelons V_{max}, l'ULMA de valeur objectif maximum atteint par un optimisation à un temps donné. Les valeurs objectif de V_{max} et V_{ref} seront respectivement notées indépendamment de la fonction utilisée par OF_{max} et OF_{ref} (nous préciserons la fonction au besoin). Si au cours d'une optimisation, le signal est retrouvé, ce qui signifie que V_{max} est identique à V_{ref}, l'optimisation est poursuivie afin de vérifier si un autre ULMA de valeur objectif supérieure peut être trouvé. L'optimisation est alors poursuivie pendant un temps CPU égal à celui qui a été nécessaire pour trouver V_{ref}, mais en gardant toujours un minimum de 30 secondes CPU pour l'optimisation totale. Si aucun point de valeur supérieure à OF_{ref} n'est trouvé pendant la poursuite de l'optimisation, nous considérons que l'ensemble de séquences contient un signal *perceptible* pour la fonction concernée, c'est à dire que ce signal se trouve au dessus du niveau du bruit. Il est clair que nous ne pouvons garantir la non-existence d'un ULMA de valeur supérieure, mais elle semble dans ce cas peu probable. Si au cours d'un optimisation, un alignement de valeur supérieure à OF_{ref} est trouvé, cela signifie que le signal est *imperceptible* et que la fonction objectif concernée n'est pas capable de l'extraire du bruit. Pour une série $\mathcal{S}_i$, la somme des signaux perceptibles et imperceptibles correspond donc à Z_i, le nombre d'ensembles de séquences de la série.

5.4.3 Comparaison des fonctions objectif

Nous avons ainsi comparé les deux fonctions objectif sur leur capacité à extraire le signal du bruit. Les optimisations sont effectuée avec la stratégie P. Pour chaque série et pour les deux fonctions, nous présentons dans les graphiques de la Figure 5.6, les taux (tx) de signaux perceptibles, calculés par le nombre d'ensembles comportant un signal perceptible, divisé par le nombre d'ensemble total de la série. Nous présentons également pour chaque série le ratio des taux de signaux détectés par $OF^{(HR)}$ sur ceux de $OF^{(H)}$. On peut constater un taux tx maximum pour les deux fonctions concernant les ensembles de $\mathcal{S}_2$, correspondant à une valeur $d_{av} = 400$. Pour les valeurs supérieures ($\mathcal{S}_3..\mathcal{S}_8$), ce taux décroît rapidement pour les deux fonctions. Il reste cependant considérablement plus élevé pour $OH^{(HR)}$, qui est capable dans le cas de $\mathcal{S}_8$ ($d_{av} = 1600$) de détecter jusqu'à sept fois plus de signaux que $OF^{(H)}$. On constate également que pour les valeurs d_{av} inférieures à 400, le nombre de signaux détectables baisse pour les deux fonctions. Ce comportement s'explique par le fait que pour ces distances évolutionnaires très faibles, les séquences sont très conservées sur toute leur longueur. Les deux fonctions attribuent alors souvent une valeur objectif supérieure à un ULMA ne correspondant pas au signal. Nous avons toutefois observé que tous les V_{max} détectés dans $\mathcal{S}_1$ étaient tout de même composés d'occurrences homologues selon l'histoire évolutionnaire de l'ensemble. Dans ce cas, les ULMAs préférés sont ceux dont la composition en acides aminés est biaisée vers des acides aminés rares, tels que spécifiés par les valeurs de fréquences de bruit de fond F^0 et R^0 utilisés respectivement par $OF^{(H)}$ et $OF^{(HR)}$. Il n'est pour ces niveaux de conservation très élevés, pas possible de trancher entre deux ULMAs homologues car le choix ne dépend plus que des valeurs de F^0 et R^0. On peut supposer que pour $d_{av} = 0$, le nombre de signaux perceptibles atteindra une valeur proche de zéro pour les deux fonctions.

Comparaisons sur le temps CPU et sur le nombre d'évaluations de OF

Nous avons ensuite comparé les deux fonctions objectif sur leur capacité à identifier un signal en fonction du temps CPU et du nombre d'évaluations nécessaires. Il est impératif pour cette expérience de comparer les deux fonctions sur les mêmes données. Nous n'avons alors considéré dans chaque série, que les ensembles de séquences dont le signal est perceptible à la fois pour les deux fonctions. Étant donné le faible nombre d'ensembles satisfaisant cette propriété dans les séries de valeur d_{av} élevée (uniquement sept ensembles sur les deux mille de $\mathcal{S}_8$), nous avons fixé une tolérance sur l'identification d'un signal. Nous avons alors retenu pour chaque série, les ensembles de séquences pour lesquels $sim(V_{max}, V_{ref}) \geq 0.75$ (la mesure de similarité sim à été définie dans la Section 5.3.3). En utilisant cette tolérance, L'ULMA V_{max} sera considéré comme correspondant à V_{ref} si :

- Il partage au moins quatorze occurrences sur seize.
- Il partage au moins quinze occurrences sur seize lesquelles peuvent être décalées d'au plus deux positions.
- Toutes les occurrences sont identifiées avec un décalage d'au plus trois positions.

Le tableau suivant donne pour chaque série, le nombre d'ensembles (#) pour lesquels le signal a été identifié par les deux fonctions à la condition précitée, ainsi que les valeurs sim moyennes d'identification du signal pour les deux fonctions ($simOF^{(HR)}$ et $simOF^{(H)}$).

	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_6$	$\mathcal{S}_7$	$\mathcal{S}_8$
#	978	995	986	824	519	210	172	59
$simOF^{(HR)}$	0.97	0.99	0.98	0.97	0.95	0.93	0.92	0.91
$simOF^{(H)}$	0.98	0.99	0.98	0.94	0.91	0.88	0.87	0.84

On peut constater que la fonction $OH^{(HR)}$ identifie les signaux avec une meilleure précision moyenne que $OF^{(H)}$. Les comparaisons sur le temps CPU et sur le nombre d'évaluations sont effectuées de la manière suivante : un taux de succès (TS) estimé pour un temps CPU ou un nombre d'évaluations donné correspond au nombre d'ensembles d'une série pour lequel le signal a été identifié à un moment donné, divisé par le nombre total d'ensembles dans la série. La Figure 5.7 présente ces résultats pour les séries $\mathcal{S}_2, \mathcal{S}_4, \mathcal{S}_6$ et $\mathcal{S}_8$, qui suffisent amplement à montrer le comportement général de ces fonctions. On constate dans tous les cas que, pour un résultat donné, le nombre d'évaluations requis est bien inférieur pour $OF^{(HR)}$. On peut interpréter cette observation par le fait que $OF^{(HR)}$ a un impact bénéfique sur les deux paysages $\mathcal{N}^{OOM}$ et $\mathcal{N}^{PS}$, rendant leur exploration par le grimpeur strict plus efficace. L'influence d'une fonction objectif sur un paysage concerne les hauteurs des noeuds qui le composent, la topologie du graphe reste par contre inchangée. Cependant, modifier ces hauteurs peut avoir des conséquences importantes sur le nombre et la disposition des maximum locaux ainsi que sur les tailles des bassins d'attraction. Le fait de devoir évaluer moins de points de l'espace de recherche pour la fonction $OF^{(HR)}$ et avec un grimpeur strict implique que la taille du bassin d'attraction du supposé maximum global est augmentée par rapport à celle produite par la fonction $OF^{(H)}$, diminuant de fait la taille moyenne des bassins des maximum locaux. Cette observation se répercute également par un avantage au niveau du temps CPU, bien que le nombre d'opérations requis pour le calcul de $OF^{(HR)}$ soit significativement plus important considérant la taille de ces données.

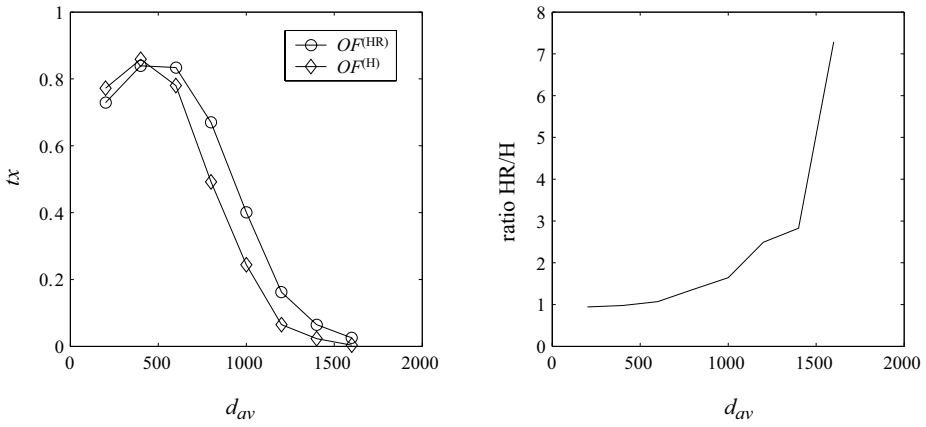


Fig. 5.6 : Les deux fonctions objectif $OF^{(HR)}$ et $OF^{(H)}$ sont comparées sur leur capacité à extraire le signal du bruit. Le graphique de gauche montre le taux de signaux perceptibles en ordonnée (tx) en fonction des valeurs d_{av} en abscisse. On constate que le taux de signal perceptible baisse drastiquement avec l'augmentation de la valeur d_{av}. Ce taux est cependant considérablement plus élevé pour la fonction $OF^{(HR)}$. Le graphique de droite montre le ratio des taux pour les deux fonctions. On constate que pour $d_{av} = 1600$ ($\mathcal{S}_8$), la fonction $OF^{(HR)}$ détecte environs sept fois plus de signaux que $OF^{(H)}$.

5.5 Alignements de motifs EF-hand

5.5.1 Le motif EF-hand

Afin d'illustrer le fonctionnement du grimpeur sur les modes ALOOPS et AOPS, nous avons constitué un ensemble de séquences protéiques contenant chacune une ou plusieurs occurrences d'un motif EF-hand (Nelson et al., 2002). Le motif EF-hand est caractéristique d'une famille de protéines ayant la propriété de lier des ions calcium (Ca++). Ce motif est très répandu dans le monde animal et il présente une conservation forte. Il est souvent présent à plusieurs exemplaires dans une même protéine. Ce motif est utilisé par des protéines impliquées dans des fonctions différentes, et on peut y voir l'illustration d'un bloc fonctionnel, conservé par l'évolution et utilisé dans différents contextes. Tous les motifs EF-hand confèrent à leur protéine la possibilité de lier un ion calcium (Ca++). La fixation ou le relâchement d'un Ca++ est spontanée et dépend de la concentration de l'ion dans le milieux cellulaire. L'utilisation de cette fonctionnalité par la protéine est de deux types. Elle peut être de type régulatrice. Dans ce cas la protéine agit comme tampon pour réguler la concentration de Ca++ libres. Le deuxième type est structurel. La liaison d'un Ca++ induit un changement de conformation de la protéine. Ce changement induit à son tour un autre événement, lequel est souvent l'activation d'une enzyme.
Le motif EF-hand est constitué de deux hélices alpha de dix à douze acides aminés séparées par une boucle de douze acides aminés. Ce type de structure est appelé "helice-boucle-hélice" ou HLH (*helix-loop-helix*). Les douze acides aminés de la partie centrale (boucle) sont directement impliqués dans la liaison avec le ion Ca++. Les résidus 1, 3, 5, 7, 9 et 12 forment une liaison chimique avec le ion. Le résidu 12, un glutamate (G) est très conservé. Les résidus 1, 3, 5, 7, et 9, peuvent différer légèrement selon la protéine et leur composition

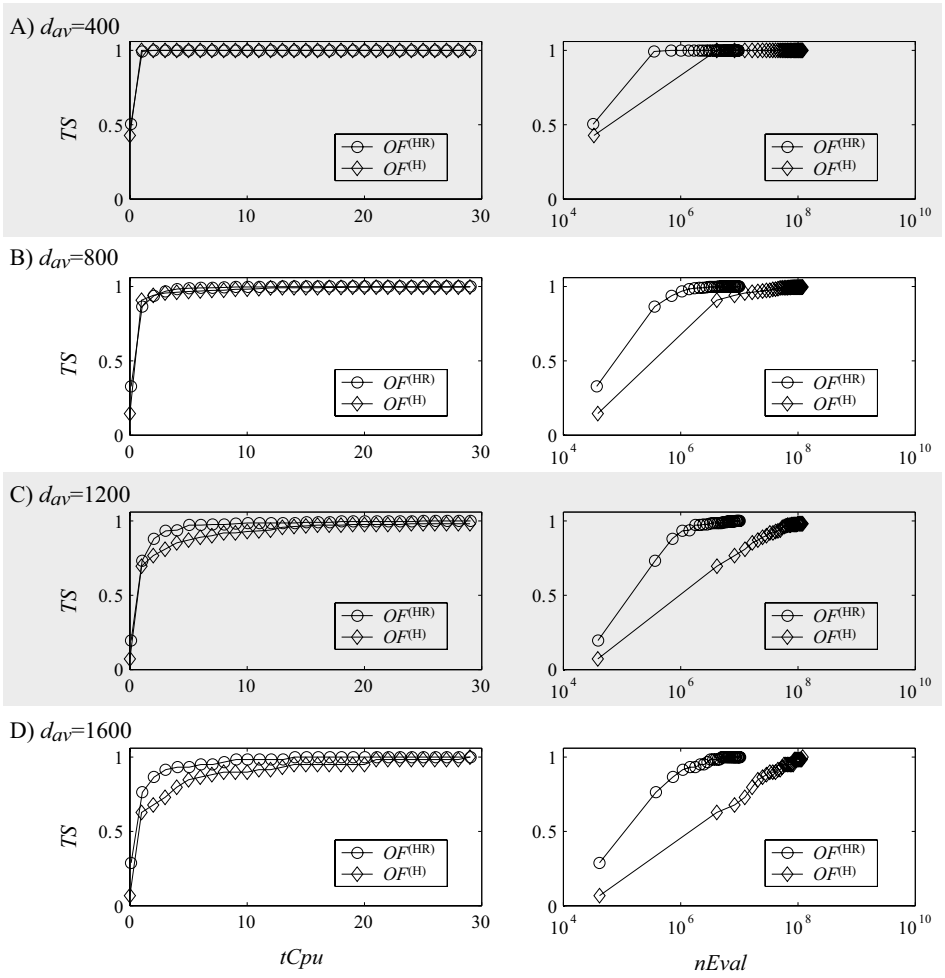


Fig. 5.7 : Quatre paires de graphiques (A,B,C,D) sont présentés, chaque paire correspondant aux valeurs d_{av} des séries $\mathcal{S}_2, \mathcal{S}_4, \mathcal{S}_6$ et $\mathcal{S}_8$. Les graphiques représentent tous en ordonnée le taux de succès (TS). Cette valeur est exprimée en fonction du temps CPU ($tCpu$) pour les graphiques de gauche, et en fonction du nombre d'évaluation ($nEval$) pour ceux de droite. Le nombre d'évaluations est donné sur une échelle logarithmique. Le temps CPU est indiqué en secondes (Athlon 1.6 Ghz). Si le temps CPU est à peu près équivalent pour les deux fonctions sur les valeurs d_{av} basses, on constate qu'il devient avantageux pour $OF^{(HR)}$ quand d_{av} augmente. Le nombre d'évaluations requis est quant à lui toujours largement inférieur pour $OF^{(HR)}$.

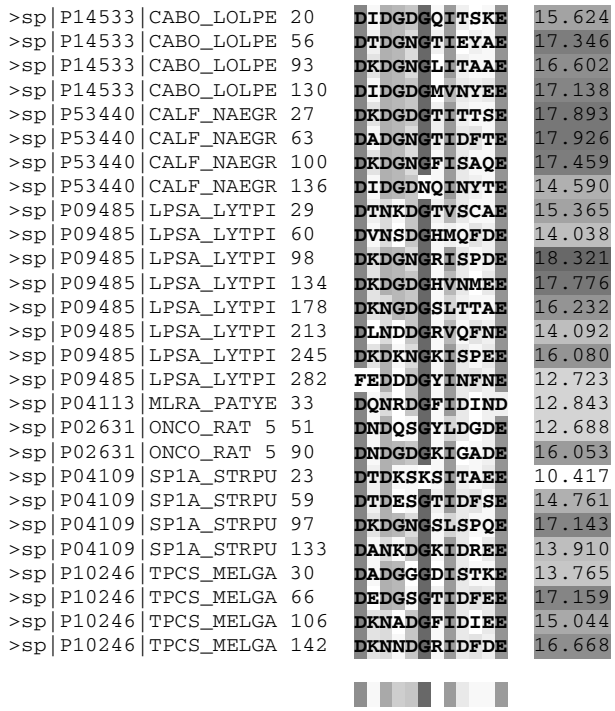


Fig. 5.8 : Alignement des domaines EF-hand. Cet alignement regroupe les vingt-sept occurrences du motif EF-hand. On peut constater que ce motif est fortement conservé.

a une implication sur l'affinité de la liaison. L'alignement que nous proposons implique ces douze résidus.

Nous avons manuellement constitué un ensemble de sept séquences protéiques contenant chacune au moins une occurrence du motif EF-hand. L'ensemble ainsi constitué comporte des séquences de longueur moyenne 173, et contient en tout 27 occurrences connues du motif. La figure 5.8 présente l'ULMA de ces occurrences, tel qu'il peut facilement être obtenu sous les modes ALOOPS et AOPS. On peut constater que les occurrences du motif sont fortement conservées et cet ensemble ne pose pas de problème d'optimisation particulier. Nous avons également constitué un deuxième ensemble, noté EF40, qui contient en plus 40 séquences aléatoires de longueur 500. Ces séquences ont été générées sur la même probabilité d'acides aminés que celle observée dans l'ensemble original (EF). Cette façon d'augmenter la difficulté que peuvent présenter des données pour un processus d'optimisation est très différente de ce que nous avons présenté dans la section 5.4 concernant le programme Rose. Le niveau de conservation des occurrences du motif EF-hand dans les données EF40 n'a pas changé. Le fait de rajouter des séquences aléatoires augmente par contre le nombre d'occurrences potentielles et certaines d'entre elles vont présenter par hasard une similarité avec le motif biologique, ou une similarité avec d'autres occurrences aléatoires. Le motif biologique étant toujours aussi conservé, la qualité de son signal pour $OF^{(HR)}$ ou $OF^{(H)}$ ne change pas, le bruit de fond est par contre augmenté. L'alignement correct peut être obtenu sur ces données avec les deux fonctions objectif.

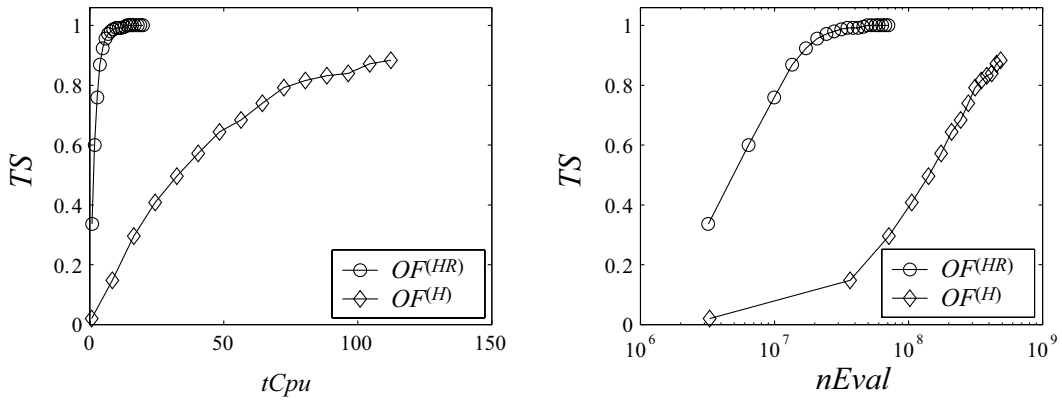


Fig. 5.9 : Graphique de performance des deux fonctions sur les données EF40. Le taux de succès (TS) est exprimé en fonction du temps CPU, et du nombre d'évaluation de la fonction objectif ($nEval$). Le nombre d'évaluations est donné sur une échelle logarithmique. On observe toujours un avantage significatif sur le nombre d'évaluations pour $OF^{(HR)}$. Sur ces données, le temps CPU bénéficie directement de cet avantage, car la taille du voisinage étant grande, le nombre d'opérations pour son évaluation est comparable entre $OF^{(HR)}$ et $OF^{(H)}$.

5.5.2 Comparaison des fonctions objectif

Nous avons effectué une comparaison des performances d'optimisation pour la stratégie P entre les deux fonctions, sur les données EF40. Le graphique de performance est présenté dans la Figure 5.9, et est obtenu par la même méthode que décrit précédemment dans la Section 5.3.2. On constate que pour un même taux de succès, $OF^{(HR)}$ est évaluée environs trente fois moins. Cependant, l'avantage de temps CPU est pour ces données considérablement supérieur à celui observé précédemment pour les données HTH ou celles générées par Rose. L'explication se trouve dans les complexités temporelles de l'évaluation du voisinage par ces deux fonctions. Comme nous l'avons montré dans la Section 4.6.1, cette complexité est de $\mathcal{O}(WU)$ pour $OF^{(H)}$, et de $\mathcal{O}(W(U+K^2))$ pour $OF^{(HR)}$, U étant la taille moyenne du voisinage. Or, il apparaît que pour ces données, $U \simeq 20100 >> K^2 = 400$, ce qui signifie que le nombre d'opérations requises pour l'évaluation d'un point de l'espace de recherche sera comparable pour les deux fonctions. L'avantage d'un nombre d'évaluations inférieur pour $OF^{(HR)}$ se traduit donc directement par un avantage sur le temps CPU.

5.5.3 Comparaison avec les méthodes classiques

Nous avons testé différentes méthodes classiques sur les données EF et EF40. Les programmes impliqués sont le Gibbs Motif Sampler (GMS) (Neuwald et al., 1995), CONSENSUS et MEME. CONSENSUS et MEME peuvent être paramétrés pour rechercher N occurrences d'un motif sous la distribution AOPS. Le GMS est un programme dédié à la recherche d'occurrences sous ce mode. On ne peut par contre pas le forcer à reporter exactement N occurrences. On doit en revanche lui fournir le nombre d'occurrences attendues qu'il va utiliser comme base pour commencer sa recherche. Les données EF ne posent aucun problème aux méthodes précitées. Elles sont toutes capables de retrouver rapidement (moins de dix secondes CPU) les vingt-sept occurrences du motif. Le résultat est par contre très variable pour les données EF40. Notre stratégie de grimpeur strict sur la

fonction $OF^{(HR)}$ retrouve comme le montre la Figure 5.9 toutes les occurrences du motif en six secondes CPU, et pour un taux de succès de 0.95. CONSENSUS retrouve toutes les occurrences du motif. Il faut pour cela augmenter la largeur du faisceau proposée par défaut avec l'option '-q 20000'. Il utilise alors environ 30 Mo de mémoire et construit l'alignement correct en vingt-deux minutes CPU. MEME ne retrouve aucune des occurrences du motif, malgré nos efforts pour optimiser ses paramètres. Le GMS ne reporte que vingt-deux occurrences, toutes correctes, en un temps CPU d'environ une seconde et pour un taux de succès de 0.95 (estimé sur deux cent optimisations indépendantes). Cette performance est tout à fait remarquable si l'on considère que ce programme ne tient pas compte de la nature des acides aminés comme le fait l'entropie recouvrante. Cette performance est due à la stratégie du GMS, qui va réduire le nombre d'occurrences qu'il considère simultanément afin de converger rapidement vers un ULMA de petite taille. Les occurrences restantes sont par la suite identifiées directement par le modèle inféré sur le petit nombre d'occurrences. Le fonctionnement du GMS est expliqué en détail dans la section 3.5.3. On peut supposer que cette stratégie est efficace pour retrouver un motif dont les occurrences sont fortement conservées, condition requise pour pouvoir identifier le signal en ne considérant qu'un petit nombre de ses occurrences (cette observation est également valable pour CONSENSUS). Il ne reporte par contre que vingt-deux occurrences sur les vingt-sept connues. Les cinq occurrences manquées ne sont probablement pas assez significatives par rapport au bruit pour qu'il puisse les identifier comme faisant partie su signal. Il faudrait en conclusion effectuer d'autres analyses afin de déterminer les limites de ce programme.

Chapitre 6

Conclusions et perspectives

6.1 Conclusions

Nous avons proposé une étude sur le problème de l'alignement local multiple et sans indels, avec une spécificité pour les séquences de protéines. Ce travail à été effectué dans un but applicatif afin de répondre à une problématique concrète. Nous pensons avoir en grande partie réussi notre pari initial, qui était le développement d'une méthode fiable pour l'optimisation d'alignements. Cette étude a abouti sur la réalisation d'un outil d'alignement qui présente comme le montre nos résultats des performances supérieures en termes d'optimisation et de reconnaissance du signal aux autres approches comparables. La liste descriptive qui suit énumère les contributions scientifiques apportées par notre travail.

- L'alignement local multiple et sans indels a été défini sous la forme d'un problème d'optimisation combinatoire par voisinage. Les définitions strictes des espaces de recherche, des fonctions de voisinage et fonctions objectif apportent une nouvelle compréhension du problème, propice à l'étude des difficultés qu'il peut présenter. Ce problème n'avait à notre connaissance pas été traité de cette façon auparavant.

- Nous avons pris en considération quatre modes de contraintes sur la répartition des occurrences. Nous les avons revus sous la forme de contraintes sur l'espace de recherche. Ces contraintes peuvent s'avérer utile à un biologiste, en fonction de ses connaissances préalables sur les données qu'il cherche à analyser.

- Une fonction objectif dédiée à l'évaluation d'alignements de séquences protéiques a été développée. Cette fonction est basée sur l'entropie relative, classiquement utilisée pour ce problème. Notre fonction garde les spécificités de l'entropie relative, mais permet en revanche de considérer la nature des acides aminés qui sont alignés. Cette fonction permet ainsi de détecter des similarités beaucoup plus faibles qu'avec la fonction classique et augmente considérablement la signification biologique des alignements produits. Elle modifie également la structure du paysage d'exploration, le rendant plus propice à une optimisation par un grimpeur. Nos expérimentations indiquent que cette fonction est à tous points de vue préférable pour l'optimisation d'alignements de séquences protéiques, par rapport à la fonction classique.

- Nous avons développé une approche consistant à générer rapidement des graines prometteuses pour le grimpeur, utilisé en conjonction avec le mode OOPS. Cette

approche permet de réduire considérablement le nombre de points de l'espace de recherche qui doivent être considérés pour permettre l'obtention d'une solution fiable. L'approche proposée ne peut pas être directement étendue aux autres modes avec la même efficacité que pour le mode OOPS. Cependant, le principe de générer rapidement des graines par assemblage direct de facteurs similaires est prometteur pour améliorer sensiblement les capacités d'exploration.

- L'implémentation de notre approche a permi la réalisation d'un programme d'alignement qui est significativement plus performant que les méthodes existantes pour ce problème. Le développement de la fonction objectif $OF^{(HR)}$, spécifique aux protéines, permet de réaliser des alignements fiables sur des séquences protéiques distantes. Ce programme a également été utilisé avec succès pour la détection de sites de régulation sur des séquences d'ADN (Yap et al., 2005).

- Nous proposons à la communauté informatique et bioinformatique une librairie de développement écrite en C++. Cette librairie permet d'intégrer directement nos stratégies d'optimisation avec leurs fonctions objectif dans un projet plus vaste. Son architecture permet de lui adjoindre rapidement des nouvelles fonctionnalités comme une fonction de voisinage ou une fonction objectif.

- Nous avons donné un point de vue sur la façon d'évaluer les méthodes. Par le biais du problème Challenge, nous avons montré que toutes les comparaisons ne sont pas pertinentes. Lors de nos comparaisons avec d'autres approches, nous avons séparé l'évaluation de la qualité des fonctions objectif, et l'évaluation de la capacité d'optimisation d'une fonction objectif donnée. Ces deux aspects sont souvent confondus et il semble primordial de les distinguer. De plus une stratégie d'optimisation combinatoire devrait toujours être évaluée en fonction du nombre de points considérés par l'exploration ainsi que du temps CPU. Ces deux points de vue sont essentiels pour pouvoir juger de la qualité globale d'une approche.

6.2 Perspectives

Cette section présente les principales perspectives que nous avons pu dégager à la fin de notre étude.

- Les stratégies Q et AG concernant le choix des graines à optimiser n'ont été développées que pour le mode OOPS. La raison était que la façon de conduire ces choix dépendait de l'opérateur de projection $MtoP$, qui effectue l'alignement d'un mot sur les séquences. Cette procédure est relativement coûteuse en calculs, et il apparaît qu'une stratégie de choix de graines n'est efficace que si les graines peuvent être produites en un temps CPU significativement inférieur à celui qui est requis par l'optimisation d'une graine par le grimpeur. Le concept général de produire des graines prometteuses apparaît très pertinent pour augmenter l'efficacité globale de la procédure d'optimisation. Des recherches devraient être menées afin d'étudier une méthode de construction de graines prometteuses efficaces du point de vue du temps CPU. Une graine prometteuse peut être simplement définie comme étant un ensemble de N occurrences de longueur W, qui présentent une similarité plus im-

portante que ce que l'on attendrait par hasard. Une piste de recherche serait de représenter S^* dans une structure de données appropriée, permettant de réunir rapidement des facteurs s^* qui présentent la propriété énoncée.

- Notre nouvelle fonction a été évaluée sur sa capacité à isoler un signal du bruit ainsi que sur l'effet qu'elle produit sur le paysage d'exploration. Une étude intéressante consisterait à évaluer le modèle probabiliste recouvrant qu'elle utilise, sur sa capacité à discriminer de nouvelles occurrences d'un motif connu. Ce travail impliquerait le développement d'une statistique sur le modèle recouvrant, permettant de décider de l'appartenance ou non d'une séquence inconnue à la famille représentée par le motif.

- Il n'existe pas à notre connaissance de comparaison complète sur la pertinence biologique des différentes fonctions objectif pour l'alignement multiple de séquences protéiques. Les études menées ne comparent pas ces fonctions sur leurs capacités à ordonner correctement les points de l'espace de recherche. Par exemple, l'étude proposée dans (Thompson et al., 2001) compare différentes fonctions, (entropie relative et somme des paires inclues), sur les écarts de valeurs qu'elles donnent entre un alignement jugé biologiquement correct et un deuxième alignement non significatif. Cette étude évalue ces fonctions sur la signification de leurs valeurs absolues, sous le point de vue d'une mesure de signifiance statistique. Bien que le fait de pouvoir distinguer un alignement aléatoire d'un alignement homologue est capital, nous pensons que ce n'est pas ce qui est demandé à une fonction objectif dédiée à l'optimisation. Le problème de la signifiance de l'alignement est statistique et doit être mesuré par la suite, sur l'alignement déjà optimisé. On attend d'une bonne fonction objectif qu'elle ait la capacité d'ordonnancer correctement les points de l'espace de recherche, les écarts de valeurs étant secondaires. Une autre propriété attendue d'une fonction objectif est la rapidité de son évaluation, dont dépend directement le temps nécessaire à une optimisation pour produire un alignement fiable. Une étude comparative, sous ce point de vue, des différentes fonctions objectif serait extrêmement intéressante.

- Une limitation majeure des programmes d'optimisation d'ULMA sont les paramètres W et N qui doivent êtres donnés par l'utilisateur. Plusieurs programmes et en particulier le Gibbs Motif Sampler, proposent une méthode permettant de lever partiellement ces contraintes. L'utilisateur doit cependant toujours donner une valeur approximative pour ces paramètres, et le programme se charge ensuite de les faire varier de façon à les optimiser. Une telle approche se fait cependant au détriment de la qualité de l'optimisation. Il serait alors utile de pouvoir évaluer une p-valeur d'un alignement, de façon à pouvoir comparer des alignements optimisés indépendamment avec différentes valeurs de N et W. Une telle approche est proposée dans (Hertz and Stormo, 1999), qui décrit une méthode analytique pour calculer la probabilité d'obtenir un score égal ou supérieur avec un alignement complètement aléatoire. Cette statistique, développée pour l'entropie relative, donne généralement de bons résultats. Elle n'est cependant pas directement applicable à notre nouvelle fonction. Il serait alors intéressant d'élaborer une telle statistique pour l'entropie relative recouvrante. Une méthode relativement fiable pour estimer une p-valeur est de procéder à un grand nombre d'optimisations sur les séquences randomisées, afin d'estimer la distribution des scores optimisés attendus sur des données non significatives (randomisées). Cette approche simple donne de bons résultats, mais elle est par contre trop

coûteuse en temps de calcul si l'on désire comparer un grand nombre d'alignements avec différentes valeurs de N de W.

- Les programmes d'optimisation d'alignement par voisinage ne permettent en général pas de considérer les événements mutationnels comme les insertions ou les délétions. Considérer ces événements lors de l'optimisation d'une fonction basée sur l'entropie pose deux problèmes. Premièrement, l'évaluation de leurs coûts, et deuxièmement l'explosion combinatoire produite. On peut imaginer un traitement des coûts d'indels par une fonction séparée et il serait alors intéressant de développer des fonctions de voisinage adaptées à ce problème, ouvrant la perspective de réaliser des alignement avec indels fiables sur des séquences distantes. Une alternative serait de limiter cette approche par voisinage à un post traitement d'un ULMA déjà obtenu, en utilisant une procédure de raffinement itératif (Wallace et al., 2005; Gotoh, 1993) afin d'y insérer des éventuels indels.

- L'optimisation simultanée de plusieurs ULMAs pourrait permettre de produire un alignement multiple complet sur les séquences. Ce concept est utilisé par MEME, qui recherche successivement un nombre donné d'ULMAs. Afin de ne pas converger deux fois sur la même solution, les occurrences déjà détectées sont masquées à chaque étape. Une piste intéressante pour permettre l'optimisation simultanée de plusieurs ULMAs est décrite par Yoann Mescam (Gras et al., 2004). Il utilise un algorithme génétique avec le concept de "niche écologique", pour permettre à plusieurs ULMAs différents de coexister pendant une optimisation.

Bibliographie

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. (2004). *Biologie moléculaire de la cellule*. Flammarion Médecine-Sciences.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *J Mol Biol*, 215 :403–410.

Avery, O., MacLeod, C., and McCarty, M. (1944). Studies on the chemical nature of the substance inducing transformation of pneumococcal types. *J. Exp. Med.*, 98 :451–460.

Bailey, T. (1995). *Discovering motifs in DNA and protein sequences : The approximate common substring problem*. PhD thesis, University of California, San Diego.

Bailey, T. L. (1993). Likelihood vs. information in aligning biopolymer sequences. Ucsd technical report cs93-318, University of California, University of California, San Diego, La Jolla, California.

Bailey, T. L. and Elkan, C. (1995). Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21(1/2) :51–80.

Baluja, S. (1996). An empirical comparison of seven iterative and evolutionary function optimization heuristics. In *Proceedings of the Eleventh International Conference on Systems Engineering*.

Barnes, J. W., Laguna, M., and Glover, F. (1995). *Intelligent Scheduling Systems*, chapter An Overview of Tabu Search Approaches to Production Scheduling Problems. Kluwer Press.

Battiti, R. and Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal on Computing*, 6 :126–140.

Blickle, T. and Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms. TIK-Report 11, TIK Institut fur Technische Informatik und Kommunikationsnetze, Computer Engineering and Networks Laboratory, ETH, Swiss Federal Institute of Technology, Gloriastrasse 35, 8092 Zurich, Switzerland.

Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.-C., Estreicher, A., Gasteiger, E., Martin, M., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S., and Schneider, M. (2003). The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Res.*, 31 :365–370.

Brazma, A., Jonassen, I., Eidhammer, I., and Gilbert, D. (1998). Approaches to the automatic discovery of patterns in biosequences. *J. Comput. Biol.*, 5(2) :277–304.

Brejova, B., DiMarco, C., Vinar, T., Hidalgo, S. R., Holguin, G., and Patten, C. (2000). Finding patterns in biological sequences. Project report for cs798g, University of Waterloo, Department of Computer Science ,Department of Statistics, Department of Biology.

Buhler, J. and Tompa, M. (2001). Finding motifs using random projections. In *Proceedings of the 5th Annual International. Conference on Computational Molecular Biology*, pages 69–76, Montreal. ACM Press.

Buhler, J. and Tompa, M. (2002). Finding motifs using random projection. *J. Comput. Biol.*, 9(2) :225–242.

Califano, A. (2000). SPLASH : structural pattern localization analysis by sequential histograms. *Bioinformatics*, 16 :341–357.

Carrillo, H. and Lipman, D. (1988). The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48(5) :1073–1082.

Carroll, S. B. (1995). Homeotic genes and the evolution of arthropods and chordates. *Nature*, 376 :479–485.

Chickering, D., Heckerman, D., and Meek, C. (1997). Learning bayesian network is NP-Hard. Technical report, Microsoft Research. MSR-TR-97-07.

Cornish-Bowden, A. (1985). Nomenclature for incompletely specified bases in nucleic acid sequences : recommendations 1984. *Nucl. Acids Res.*, 13 :3021–3030.

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978). A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5 :345–352.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis.* Cambridge University Press.

Feng, D. and Doolittle, R. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol.*, 25(4) :351–60.

Fogel, L., Owens, A., and Walsh, M. (1966). *Artificial Intelligence Through Simulated Evolution.* Chichester : Wiley.

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2004). *Basesian Data Analysis*, chapter 11-12. Chapman & Hall/CRC.

Glover, F. (1986). Future path for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5 :533–549.

Goldberg, D. and Segrest, P. (1987). Finite markov chain analysis of genetic algorithms. In Grefenstette, J., editor, *Proceedings of the Second International Conference on Genetic Algorithms.*

Goldberg, D. E. (1989). *Genetic Algorithms in Search. Optimisation and Machine Learning.* Addison-Wesley.

Gonnet, G., Cohen, M., and Benner, S. (1992). Exhaustive matching of the entire protein sequence database. *Science*, 256(5062) :1443–1445.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *J Mol Biol.*, 162(3) :705–708.

Gotoh, O. (1993). Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Computer Application in the Biosciences*, 9 :361–370.

Gras, R. (2004). Structure des espaces de recherches, complexité des algorithmes d'optimisation combinatoire stochastiques et applications à la bioinformatique. Habilitation à Diriger les Recherches, IRISA, Université de Rennes, France.

Gras, R. (2005). How efficient are evolutionary algorithms to solve high epistasie deceptive problem ? *Computer and Operations Research.* submitted.

Gras, R., Hernandez, D., Hernandez, P., Zangger, N., Mescam, Y., Frey, J., Martin, O., Nicolas, J., and Appel, R. D. (2004). *Artificial intelligence methods and tools for systems biology*, chapter Cooperative Metaheuristics for Exploring Proteomic Data, pages 87–106. Springer.

Griffith, F. (1928). The significance of pneumococcal types. *Journal of Hygiene*, 27 :113–159.

Griffiths, A. J. F., Miller, J. H., Suzuki, D. T., Lewontin, R. C., and Gelbart, W. M. (2002). *Introduction à l'analyse génétique*. De Boeck Université.

Hao, J., Galinier, P., and Habib, M. (1999). Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2) :283–324.

Henikoff, J., Greene, E., Pietrokovski, S., and Henikoff, S. (2000). Increased coverage of protein families with the blocks database servers. *Nucleic Acids Res.*, 18(1) :228–230.

Henikoff, S. and Henikoff, J. (1991). Automated assembly of protein blocks for database searching. *Nucleic Acids Res*, 19 :6565–6572.

Henikoff, S. and Henikoff, J. (1992). Amino acid substitution matrices from protein blocks. In *Proc. Natl. Acad. Sci.*, volume 89, pages 10915–10919.

Hernandez, D., Gras, R., and Appel, R. (2002). MoDEL : inférence de motifs avec un algorithme évolutionniste. In *Journée Ouverte Biologie Informatique Mathématique (JOBIM)*, pages 265–267, Saint-Malo, France.

Hernandez, D., Gras, R., and Appel, R. (2004). MoDEL : an efficient strategy for ungapped local multiple alignment. *Computational Biology and Chemistry*, 28(2) :119–128.

Hertz, G. Z., Hartzell, G. W., and Stormo, G. D. (1990). Identification of consensus patterns in unaligned DNA sequences known to be functionally related. *CABIOS*, 6(2) :81–92.

Hertz, G. Z. and Stormo, G. D. (1999). Identifying DNA and protein pattern with statistically significant alignments of multiple sequences. *Bioinformatics*, 15 :563–577.

Higgins, D., Bleasby, A., and Fuchs, R. (1992). CLUSTAL V : improved software for multiple sequence alignment. *CABIOS*, 8 :189–191.

Higgins, D., Thompson, J., and Gibson, T. (1996). Using CLUSTAL for multiple sequence alignments. *Methods in Enzymology*, 266 :383–402.

Holland, J. (1975). *Adaptation in natural and artificial systems*. The university of Michigan Press, Ann Arbor.

Horton, P. (2001). Tsukuba BB : A branch and bound algorithm for local multiple alignment of dna and protein sequences. *J. Comput. Biol.*, 8(3) :283–303.

Hughes, J. D., Estep, P. W., Tavazoie, S., and Church, G. M. (2000). Computational identification of cis-regulatory elements associated with groups of functionally related genes in saccharomyces cerevisiae. *J. Mol. Biol.*, 296(5) :1205–1214.

Jonassen, I. (1997). Efficient discovery of conserved patterns using a pattern graph. *CABIOS*, 13(5) :509–522.

Jonassen, I., Collins, J. F., and Higgins, D. G. (1995). Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4 :1587–1595.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, New Mexico.

Keich, U. and Pevzner, P. A. (2002). Finding motifs in the twilight zone. *Bioinformatics*, 18(10) :1374–1381.

Kimura, M. (1983). *The Neutral Theory of Molecular Evolution*. Cambrigde University Press.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220 :671–680.

Koza, J. (1992). *Genetic Programming : on the programming of computers by means of natural selection*. The MIT Press.

Kullback, S. (1968). *Information theory and statistics*. Dover Publications, New York.

Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C. (1993). Detecting subtle sequence signals : A gibbs sampling strategy for multiple alignment. *Science*, 262 :208–214.

Lawrence, C. E. and Reilly, A. A. (1990). An expectation maximisation (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *PROTEINS : Structure, Function, and Genetics*, 7 :41–51.

Lipman, D., Altschul, S., and Kececioglu, J. (1989). A tool for multiple sequence alignment. In *Proc. Natl. Acad. Sci.*, volume 86, pages 4412–4415.

Liu, X., Brutlag, D., and Liu, J. (2001). BioProspector : Discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In *Proceedings of Pacific Symposium on Biocomputing (PSB 2001)*, pages 127–138.

Marsan, L. (2002). *Inférence de motifs structurés : algorithmes et outils appliqués à la détection de sites de fixation dans des séquences génomiques*. PhD thesis, Université de Marne-la-Vallée, 77454 Marne la Vallée FRANCE.

Marsan, L. and Sagot, M.-F. (2000). Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification. *J. Comput. Biol.*, 7 :345–360.

Martin, O., Gras, R., Hernandez, D., and Appel, R. D. (2003). Optimizing genetic algorithms using self-adaptation and explored space modelization. In *fifth International Workshop on Frontiers in Evolutionary Algorithm*, pages 291–294, North Carolina.

Mirny, L. and Shakhnovich, E. (1999). Universally conserved positions in protein folds : reading evolutionary signals about stability, folding kinetics and function. *J. Mol. Biol.*, 291 :177–196.

Morgenstern, B. (1999). DIALIGN 2 : improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3) :211–218.

Morgenstern, B., Dress, A., and Werner, T. (1993). Multiple DNA and protein sequence alignment based on segment-to-segment comparison. In *Proc. Natl. Acad. Sci.*, volume 93, pages 12098–12103.

Muhlenbein, H. and Mahnig, T. (2001). *Theoretical Aspects of Evolutionary Computing*, chapter Evolutionary Algorithm : From Recombination to Search Distribution, pages 135–173. L. Kalled, B. Naudts, A. Rogers, eds Springer.

Mühlenbein, H. and Voigt, H.-M. (1995). Gene pool recombination in genetic algorithms. In *Proceedings of the International Meta-Heuristics Conference*, Norwell. Kluwer Academic Publishers.

Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3) :443–53.

Nelson, M. R., Thulin, E., Fagan, P. A., Forsén, S., and Chazin, W. J. (2002). The EF-hand domain : A globally cooperative structural unit. *Protein Science*, 11 :198–205.

Neuwald, A. F., Liu, J. S., and Lawrence, C. E. (1995). Gibbs motif sampling : Detection of bacterial outer membrane protein repeats. *Protein Science*, 4 :1618–1632.

Nielsen, M., Lundegaard, C., Worning, P., Hvid, C., Lamberth, K., Buus, S., Brunak, S., and Lund, O. (2004). Improved prediction of MHC class I and class II epitopes using a novel gibbs sampling approach. *Bioinformatics*, 20(9) :1388–1397.

Notredame, C. (2002). Recent progress in multiple sequence alignments. *Pharmacogenomics*, 3(1) :131–144.

Notredame, C., Higgins, D., and Heringa, J. (2000). T-Coffee : A novel method for multiple sequence alignments. *J. Mol. Biol.*, 302 :205–217.

Notredame, C., Holme, L., and Higgins, D. (1998). COFFEE : a new objective function for multiple sequence alignmnent. *Bioinformatics*, 14(5) :407–422.

O'Sullivan, O., Suhre, K., Abergel, C., Higgins, D., and Notredame, C. (2004). 3DCoffee : combining protein sequences and structures within multiple sequence alignments. *J. Mol. Biol*, 340 :385–395.

Pelikan, M. (2002). *Bayesian Optimization Algorithm : From Single Level to Hierarchy*. PhD thesis, University of Illinois.

Pelikan, M. and Goldberg, D. (2001). Escaping hierarchical traps with competent genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO*, pages 511–518, San Francisco, California.

Pelikan, M., Goldberg, D., and Cantu-Paz, E. (1999). BOA : the bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO*, pages 525–532, Orlando, Florida.

Pevzner, P. and Sze, S.-H. (2000). Combinatorial approachs to finding subtle signals in DNA sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, volume 8, pages 269–278, San Diego. AAAI Press.

Price, A., Ramabhadran, S., and Pevzner, P. A. (2003). Finding subtle motifs by branching from sample strings. *Bioinformatics*, 19(suppl 2) :149ii–155.

Rechenberg, I. (1973). *Evolutionsstrategie : Optimierung Techiquer Systeme nach Prinzipien der Biologischen Evolution*. Stuttgart : Frommann-Holzboog verlag.

Rigoutsos, I. and Floratos, A. (1998). Combinatorial pattern discovery in biological sequences : The TEIRESIAS algorithm. *Bioinformatics*, 14(1) :55–67.

Rosinski, J. A. and Atchley, W. R. (1999). Molecular evolution of helix-turn-helix proteins. *J. Mol. Evol.*, 49 :301–309.

Russel, S. and Norvig, P. (2003). *Artificial Intelligence : A Modern Approach*. Prentice Hall.

Saitou, N. and Nei, M. (1987). The neighbor-joining method : a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4 :406–425.

Sauder, J. M., Arthur, J. W., and Dunbrack, R. L. (2000). Large-scale comparison of protein sequence alignment algorithms with structure alignments. *Proteins*, 40 :6–22.

Schmid, C., Praz, V., Delorenzi, M., Perier, R., and Bucher, P. (2004). The eukaryotic promoter database EPD : the impact of in silico primer extension. *Nucleic Acids Res.*, 32. Database issue D82-D85.

Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423, 623–656.

Sharpe, O. J. (2000). *Towards a Rational Methodology for Using Evolutionary Search Algorithms*. PhD thesis, University of Sussex, Brighton, UK.

Sinha, S. and Tompa, M. (2002). Discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucl. Acids Res.*, 30 :5549–5560.

Smith, H., Annau, T., and Chandrasegaran, S. (1990). Finding sequence motifs in groups of functionally related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 87(2) :826–830.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *J Mol Biol*, 147 :195–197.

Stoye, J., Evers, D., and Meyer, F. (1998). Rose : generating sequence families. *Bioinformatics*, 14 :157–163.

Stoye, J., Moulton, V., and Dress, A. (1997). DCA : an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. *CABIOS*, 13(6) :625–626.

Thijs, G., Lescot, M., Marchal, K., Rombauts, S., Moor, B., Rouzé, P., and Moreau, Y. (2001). A higher-order background model improves the detection of promoter regulatory elements by gibbs sampling. *bioinformatics*, 17(12) :1113–1122.

Thompson, J., Higgins, D., and Gibson, T. (1994). CLUSTAL W : improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22 :4673–4680.

Thompson, J., Plewniak, F., Ripp, R., Thierry, J., and Poch, O. (2001). Towards a reliable objective function for multiple sequence alignments. *J. Mol. Biol*, 314 :937–951.

Thompson, W., Rouchka, E. C., and Lawrence, C. E. (2003). Gibbs recursive sampler : finding transcription factor binding sites. *Nucleic Acids Research*, 31(13) :3580–3585.

Valdar, W. (2002). Scoring residue conservation. *PROTEINS : Structure, Function, and Genetics*, 48 :227–241.

van Helden, J., André, B., and Collado-Vides, J. (1998). Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J. Mol. Biol.*, 281(5) :827–842.

Vingron, M. and Waterman, M. (1994). Sequence alignment and penalty choice. review of concepts, case studies and implications. *J Mol Biol*, 235(1) :1–12.

Wallace, I. M., O'Sullivan, O., and Higgins, D. G. (2005). Evaluation of iterative alignment algorithms for multiple alignment. *Bioinformatics*, 21(8) :1408–1414.

Watson, J. and Crick, F. (1953). Molecular structure of nucleic acids : a structure for deoxyribose nucleic acid. *Nature*, 171 :737–738.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transaction on Evolutionary Computation*, 1(1) :67–82.

Yap, Y. L., Lam, D. C. L., Girard, L., Zhang, X. W., Hernandez, D., Gras, R., Wang, E., Chiu, S. W., Chung, L. P., Lam, W. K., Smith, D. K., Minna, J. D., Danchin, A., and Wong, M. P. (2005). Conserved transcription factor binding sites of cancer markers derived from primary lung adenocarcinoma microarrays. *Nucleic Acids Res.*, 33(1) :409–421.

Zhou, Q. and Liu, J. S. (2004). Modeling within-motif dependence for transcription factor binding site predictions. *Bioinformatics*, 20(6) :909–916.