# UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

**Formation Doctorale : Contrôle des Systèmes**

# THÈSE DE DOCTORAT

*pour obtenir le grade de Docteur de l'Université de Technologie de Compiègne*

Discipline : Informatique

*présentée par*

**Jean-Paul COMET**

---

# Programmation Dynamique et Alignements de Séquences Biologiques

---

Soutenue le 16 novembre 1998
devant le jury composé de :

| | | | |
|---|---|---|---|
| M. | Maxime | CROCHEMORE | Rapporteur |
| M. | Bernard | PRUM | Rapporteur |
| M. | Dominique | CELLIER | Examinateur |
| M. | Gérard | GOVAERT | Examinateur |
| M. | Jean-Eric | PIN | Examinateur |
| M. | Jean-Loup | RISLER | Examinateur |
| M. | Pierre | VILLON | Examinateur |
| M. | Jacques | HENRY | Directeur |

# Remerciements

Je tiens ici à exprimer toute ma reconnaissance à *Jacques Henry* pour la façon avec laquelle il m'a dirigé tout au long de ma thèse. Son écoute a toujours été d'un grand réconfort lorsque le découragement ou simplement la lassitude pointait à l'horizon. A ces moments-là, il a toujours su me relancer sur une voie légérement différente, et mon travail n'en a été que plus intéressant. Je tiens à le remercier pour m'avoir initier au metier de chercheur.

*Bernard Prum* et *Maxime Crochemore* ont accepté d'être les rapporteurs de ce manuscrit en dépit du peu de temps dont ils disposent. Qu'ils veuillent bien trouver ici l'expression de ma vive reconnaissance.

Je remercie *Dominique Cellier*, *Gérard Govaert*, *Jean-Eric Pin* et *Pierre Villon* pour avoir chacun accepté d'être membre du jury. *Jean-Loup Risler* a été tout au long de cette thèse d'une aide précieuse puisqu'il a été mon interlocuteur de tous les jours pour la biologie. Je ne peux que me réjouir de sa présence au sein du jury.

*Jean-Jacques Codani* a également beaucoup contribué au bon déroulement de cette thèse en en suivant attentivement les progressions. C'est à lui que reviennent mes remerciements pour le travail d'équipe dans laquelle je me suis inséré. La coordination du projet n'était pas chose facile tant les compétences et les aspirations de chacun étaient différentes.

Je remercie vivement *Eric Glémet* qui m'a été d'un grand secours dès mon arrivée puisqu'il m'a permis de me confronter rapidement aux difficultés que j'allais rencontrer. Eric m'a formé à l'utilisation du logiciel LASSAP et à sa programmation, ce qui m'a permis d'effectuer des tests à grande échelle.

Les travaux de *Jean-Christophe Aude* ont été par moments très voisins des miens. Il s'en est suivi un travail en commun dont les résultats ont particulièrement intéressé les biologistes.

Je ne peux oublier le Professeur *Piotr Slonimski*, qui a été un élément à part entière de l'équipe. Ses compétences et son expérience dans le domaine de la biologie moléculaire nous ont particulièrement portés et les résultats n'auraient pas autant de pertinence sans son regard de chercheur en biologie.

Je tiens à remercier *Jean-Noël Bacro*, *Bernard Prum*, *Sophie Schbath* et *Andrzej Wozniak* pour leurs encouragements et leurs aides.

Je remercie tous ceux qui ont relu ce manuscrit et qui par leur remarques judicieuses ont contribué à l'élaboration de sa version finale.

*Life can only be understood going backwards,*
*but it must be lived going forwards.*

*Søren Kierkegaard (1813-1855)*

# Introduction

Dans tout organisme vivant, on trouve deux familles de macromolécules ou polymères qui sont particulièrement étudiées. Il s'agit des séquences d'acides nucléiques et des séquences d'acides aminés. Les séquences d'acides nucléiques ont pour principale fonction de stocker l'information génétique. L'information héréditaire est portée par une ou plusieurs molécules d'acide désoxyribonucléique, l'**ADN**. L'acide ribonucléique, l'**ARN**, intervient, lui, dans le processus de la synthèse des protéines.

Les protéines, séquences d'acides aminés, forment la seconde famille de macromolécules et sont les principaux acteurs des différents mécanismes cellulaires : catalyse, transport d'autres molécules, transmission de messages...

La détermination et la compréhension de la fonction des protéines au sein d'une cellule est un des enjeux majeurs de la biologie actuelle. Les méthodes expérimentales directes permettant d'identifier la fonction d'une protéine donnée sont très lourdes à mettre en œuvre, et l'expérimentation sur l'être humain pose des problèmes d'éthique. De plus, depuis quelques années le séquençage s'est banalisé permettant une croissance exponentielle des banques de données. Mais en même temps, on observe une diminution du pourcentage de séquences sur lesquelles des informations biologiques ou physiques directes sont disponibles. Aussi, des méthodes d'analyse procédant par analogie ont été développées. Elles permettent de tirer des principes généraux à partir de la comparaison des séquences entre elles, afin de déterminer la fonction d'une protéine à partir de sa structure chimique ou physique.

Pour faire face à ces enjeux et aux prochains défis de la biologie (la compréhension de l'information génétique et la guérison par modification de l'information génétique), une nouvelle discipline a émergé au sein de la communauté scientifique internationale : la **bio-informatique**, regroupant trois profils de chercheurs : biologistes, informaticiens et mathématiciens.

La bio-informatique tente d'apporter des réponses aux différentes questions suivantes :

- ***Acquisition de séquences ou séquençage, contigs :*** cette phase a pour but l'acquisition de la structure dite *primaire* de la séquence, c'est-à-dire la suite des acides nucléiques. Malgré l'apparition de nouvelles méthodes automatiques de séquençage, le séquençage de brins d'ADN pose quelques problèmes. Les deux stratégies possibles utilisent le séquençage de sous-fragments chevauchants [73, 74]. La méthode directe (Primer Walking) parcourt la séquence pas à pas. La méthode aléatoire (shotgun) génère aléatoirement un grand nombre de sous-fragments chevauchants, et ces sous-fragments sont assemblés en *contigs* (contiguous longer sub-fragments). Les differents contigs ne recouvrent pas forcément la totalité de la séquence et l'ordre des contigs reste inconnu.

- ***Comparaison de séquences :*** la majorité des nouvelles séquences provient du séquençage d'ADN, et non de protéines. Cependant, la comparaison de séquences sur les protéines est

plus fine. A cela deux raisons. La première est la dégénérescence du code génétique (Cf. annexe A). Dans le cas de l'acide aminé appelé *serine*, un codon **UCA** (Ser) sera aligné par trois substitutions non conservatives lorsqu'il sera comparé au codon **AGU** (Ser). Une fois transcrit en protéines, l'égalité des acides aminés est triviale. D'autre part, dans le cas des protéines, les éléments de base (les acides aminés) sont au nombre de 20. Les métriques possibles sur l'ensemble des acides aminés sont donc plus nombreuses que dans le cas des acides nucléiques où il n'y a que 4 éléments de base et, par conséquent, peuvent être plus fines. Cette notion de similitude entre deux acides aminés est généralement donnée par une matrice de similarité dont on peut trouver un exemple en annexe A, figure A.4.

- ***Détermination de la structure secondaire voire tertiaire :*** bien que la structure primaire des protéines (la suite des acides aminés) soit importante, la structure tridimensionnelle donne beaucoup plus d'information sur la fonction de la protéine. Les méthodes expérimentales de visualisation des structure tertiaires (cristallographie aux rayons X, résonance magnétique nucléaire) sont trop coûteuses en temps pour être généralisées. Puisque l'écart entre le nombre de séquences primaires connues et le nombre de structures connues augmente sans cesse, et que la structure tertiaire permet une meilleure connaissance de la macromolécule, la recherche de solutions calculatoires pour la détermination des structures devient un domaine utile qui met l'accent sur les relations entre la séquence primaire et tertiaire.

- ***Phylogénie :*** il s'agit de la construction d'arbres représentant l'évolution des espèces. Ces arbres sont souvent sans racine, puisqu'il est souvent impossible d'identifier l'origine du processus de diversification. On possède maintenant suffisamment de données génétiques pour pouvoir vérifier ou infirmer les hypothèses faites sur l'évolution des êtres vivants.

  Il y a trois méthodes générales pour la construction d'arbres phylogénétiques, comportant chacune de nombreuses variantes : la parcimonie, les matrices de distances et les méthodes de maximum de vraisemblance.

  - Les méthodes parcimonieuses : ces méthodes [24], qui ne font pas de supposition sur les relations entre les organismes a priori, essayent simplement de déterminer l'arbre le plus parcimonieux parmi tous les arbres possibles, c'est-à-dire celui qui minimise le nombre total d'événements mutationnels. L'idée de base est de faire croître un arbre par ajout successif d'espèces, puis éventuellement de retoucher l'arbre parcimonieux par réarrangement local.
  - Les méthodes de matrices de distance : à partir de la donnée d'une matrice de distance [1], on cherche à reconstruire une distance évolutive, qui représente le nombre d'événements mutationnels réels et qui est une distance arborée. On peut citer le programme ADDTREE [109], qui grâce à un algorithme glouton construit un arbre rapidement en remplaçant deux éléments par leur barycentre. On peut aussi citer les développements de Saitou et Nei [108], dont la méthode est certainement la plus utilisée.

    La méthode la plus représentative de cette classe est certainement celle de Sokal et Sneath [118] : **UPGMA** pour *Unweighted Pair Group Mean Analysis.*
  - Les méthodes du maximum de vraisemblance : ces méthodes ont été introduites par Edwards et Cavalli-Sforza [37] et largement étendues par Felsenstein [40]. On se donne

1. En règle générale, il ne s'agit pas d'une distance au sens mathématique du terme, puisque l'inégalité triangulaire n'est pas vérifiée.

un modèle stochastique de l'évolution comme ceux de Jukes et Cantor [77] ou Kimura [81].

- ***Classification :*** La classification de séquences est un problème proche de la phylogénie. Il ne s'agit plus forcement de classer des organismes entre eux, mais par exemple de classer les séquences protéiques d'un même organisme. On peut rajouter aux méthodes provenant de la phylogénie toutes les approches classiques d'analyse de données, en particulier les pyramides [20], dont l'application au cas des séquences biologiques est fructueuse [9].

***La comparaison de séquences ainsi que l'alignement*** sont des sujets qui posent des problèmes riches et intéressants pour l'informatique, que ce soit du point de vue théorique ou pratique [113]. Le problème générique s'exprime très simplement. Etant données deux suites d'éléments, trouver un algorithme, qui détermine les deux sous-suites, qui sont, par rapport à une certaine notion de similarité entre sous-suites, celles qui maximisent ce critère de similarité.

Ce type de problème se retrouve dans beaucoup d'applications dans des domaines qui paraissent vraiment très éloignés, comme l'informatique, les mathématiques, la biologie moléculaire, la reconnaissance de la parole et bien d'autres encore... Voici différentes applications :

- En *Informatique*, ce problème a au moins deux applications, dont la plus largement utilisée est la comparaison de fichiers : la commande **diff** UNIX [75] permet de rechercher les différences (par ligne) entre les deux fichiers. Dans cette application, l'élément de base est une ligne complète. Le but est de trouver la sous-séquence commune de longueur maximale, c'est-à-dire une suite de lignes, dont chaque ligne apparaît dans les deux fichiers dans le même ordre. Ce problème est souvent appelé *lcs* pour *Longest Common Subsequence* [67, 90, 129]. La deuxième application en informatique est la détection des erreurs de frappe, pour les correcteurs orthographiques. Il s'agit là de la recherche des mots les plus proches dans un dictionnaire.

- Les applications à la *Linguistique* sont nombreuses [25][22][45]. A partir d'un corpus bilingue, on cherche à aligner les phrases puis les mots, c'est-à-dire qu'on cherche à mettre en correspondance des entités linguistiques d'une langue en face de celles de l'autre langue. L'alignement de phrases est la première phase indispensable pour la création de dictionnaires probabilistes, et pour la traduction automatique. La programmation dynamique est une méthode classique bien adaptée à ce type de problèmes qui permet de décomposer le problème en sous-problèmes. Mais puisqu'on se permet de traduire une phrase par deux phrases et inversement, la relation de récurrence n'est pas aussi simple que dans le cas où chaque entité est traduite par une et une seule entité. Les coûts d'alignements sont calculés statistiquement sur le corpus d'apprentissage.

- La *Biologie* a permis d'élargir la problématique, surtout dans la cadre de la comparaison de **protéines** ou de chaînes d'**ADN**. La métrique associée à l'alphabet n'est pas triviale. Pour les méthodes de *string matching*, deux cas se présentaient : deux lettres sont soit identiques soit différentes. Les biologistes ont introduit une métrique sur les éléments de bases : les acides aminés (pour les protéines). Cette métrique est symétrique, réprésentée par une matrice (Cf. 1.1.4). Le problème de la recherche de la plus longue sous-séquence commune se généralise ici pour mettre en évidence les deux sous-séquences qui semblent, suivant la métrique citée plus haut, être les deux plus proches. On ne s'intéresse plus directement à la globalité des données mais à trouver les régions qui se ressemblent.

Le travail présenté ici porte sur les algorithmes de comparaison de séquences biologiques. Si on se restreint au champ d'application de la biologie moléculaire, la recherche d'un alignement de deux séquences de coût minimal[2] se résout facilement par un algorithme de **Programmation Dynamique**. Cet algorithme est de complexité quadratique si les coûts d'insertion/délétion sont affines ou linéaires. Ce même algorithme peut s'exprimer de manière simple dans l'algèbre $(max, +)$. Cette approche permet de construire un automate pour la recherche de similarité locale ou globale. Malheureusement cet automate est de taille énorme, mais il permettrait le parcours d'une banque de données en temps linéaire.

L'algorithme de programmation dynamique ne prend en compte que la structure primaire des deux séquences. Il en est de même pour la majorité des programmes d'alignement de séquences. Le biologiste, lorsqu'il aligne deux séquences, prend en compte d'autres types d'information : comme les motifs qu'il sait être pertinents pour les séquences considérées. Cherchant à améliorer les alignements obtenus par programmation dynamique, nous proposons une méthode d'alignement prenant en compte des informations autres que la structure primaire. Par exemple, la prise en compte des *motifs* contenus dans la banque *PROSITE* peuvent permettre d'améliorer les alignements. L'idée de base est très simple : favoriser un alignement, si celui-ci met en correspondance les occurrences du même motif présentes dans les deux séquences.

Les questions d'ordre statistique apparaissent naturellement, lorsque le chercheur veut mettre en évidence des relations entre séquences, non encore établies. Bien que la signification statistique ne soit ni nécessaire ni suffisante pour en déduire une signification biologique, elle reste cependant un bon indicateur. La signification statistique est d'autant plus utile que le nombre de comparaisons à effectuer (toute nouvelle séquence contre toutes celles déjà connues) augmente très vite en raison de la croissance exponentielle des banques de données. La signification statistique peut alors aider à rejeter la majorité des comparaisons et faire ressortir, au milieu du flot de résultats, des relations biologiques non triviales.

On peut classer les alignements non pas par score, mais par leur *significativité*. La significativité associée à un alignement de score $s$ représente la probabilité d'observer un score au moins égal à $s$. Les modèles théoriques existants ne s'appliquent pas directement au cas de la programmation dynamique. Nous proposons une évaluation de la significativité du score par simple permutation des séquences. Le *Z-score* est le score centré - réduit par rapport à un échantillon de scores obtenus avec des permutations des séquences initiales.

Il apparaît que le *Z-score* tend à éliminer le biais dû à des compositions en acides aminés différentes et à des longueurs différentes. Cependant, il est nécéssaire de modéliser le comportement du *Z-score*. Une étude sur des génomes complets fait apparaître un modèle de Gumbel, modèle théorique sous-jacent à la recherche de similarité sans insertions/délétions (BLAST).

**Plan de lecture**

Dans le premier chapitre, nous rappellerons les principaux algorithmes pour aligner deux séquences. Historiquement, le premier à avoir vu le jour est celui de Needleman et Wunsch, puis de Smith & Waterman, suivi de près par les heuristiques les plus utilisées que sont BLAST et FASTA.

Le second chapitre s'intéresse plus précisément aux algorithmes de Needleman & Wunsch et de Smith & Waterman. Quelques propriétés simples et très intuitives sont mises en évidence. Quelques-unes permettent de justifier certaines accélérations de l'algorithme, et une implémentation parallèle

2. Le coût de l'alignement correspond au coût d'édition.

de l'algorithme de Smith & Waterman grâce aux instructions graphiques.

La troisième partie reformule les algorithmes précédents dans l'algèbre (max,+). Cette formulation permet de concevoir un automate pour la recherche de l'alignement optimal par programmation dynamique. Même si cet automate est dans bien des cas impossible à construire, à cause de sa taille, cette vision de la programmation dynamique fait le lien entre différents domaines de l'informatique : le *pattern-matching* avec erreurs et l'alignement de séquences.

Nous introduisons au cours de la quatrième partie un algorithme hybride provenant de l'algorithme de Smith & Waterman prenant en compte un autre type d'information, la présence dans les deux séquences d'une occurrence du même motif *PROSITE*. Cet algorithme est encore de la classe de la programmation dynamique, et a été testé sur un grand nombre de données.

Enfin, la significativité d'un score d'alignement a été un objectif tout au long de ce travail. Le Z-score est un bon guide statistique pour l'évaluation de la significativité biologique d'un alignement. Il repose sur processus de Monte-Carlo, afin d'essayer de tenir compte de la dépendance du score en fonction de la longueur et en fonction de la composition en acides aminés. Le *Z-score* apparaît être un indicateur un peu plus fiable que le score.

# Chapitre 1

# Comparaison de séquences : l'état de l'art

*Avec le progrès des techniques de séquençage, le nombre de séquences répertoriées dans les bases de données spécialisées croit de façon fulgurante, en même temps que l'information disponible sur ces nouvelles séquences s'appauvrit en se réduisant à la structure primaire de la séquence. Le biologiste cherche donc le plus souvent à analyser une séquence à partir de la seule connaissance de sa structure primaire en la comparant à des bases très grandes. L'objectif est d'essayer d'induire des connaissances sur cette séquence en utilisant une grande masse de données.*

*La comparaison de séquences d'***ADN** *ou de* **protéines** *est un outils très utile pour discerner la fonction, la structure ou pour établir des relations d'évolution. La suite des acides aminés, dans une protéine détermine en partie la forme tridimensionnelle de la protéine, qui à son tour, influence la fonction de la protéine. Les sites actifs tendent à être conservés et ainsi, on peut trouver des similitudes entre les protéines portant les mêmes fonctions. Ceci explique tout l'intérêt de découvrir les régions similaires entre deux protéines.*

*Les méthodes de comparaison tentent toutes de résoudre un problème de minimisation d'une fonction* coût. *Lorsque la fonction* coût *est simple, la programmation dynamique résout efficacement le problème, mais, reste un algorithme coûteux en temps. Certaines heuristiques ont alors été introduites et proposent de réduire l'espace de recherche, pour diminuer le temps de calcul.*

*L'alignement multiple, qui cherche à mettre en correspondance les zones similaires d'un ensemble de séquences sera abordé. La généralisation de la programmation dynamique au problème de l'alignement multiple mène à des algorithmes dont le temps de calcul croît de manière exponentielle avec le nombre de séquences. D'autres approches ont été proposées afin de réduire le temps de calcul, mais elles ne donnent que des solutions sous-optimales. Nous donnerons les principes des heuristiques les plus utilisées.*

## 1.1 Définitions et notations

### 1.1.1 Alphabets et séquences

Un alphabet A est un ensemble de symboles, à partir duquel seront écrit des mots. En biologie moléculaire, on considère deux alphabets différents :

– l'alphabet des acides nucléiques sur lequel seront retranscrites les séquences d'ADN : $\mathcal{A} = \{A, C, G, T\}$, $A$ pour Adénine, $C$ pour Cytosine, $G$ pour Guanine et $T$ pour Thymine. D'une

manière similaire, l'alphabet de l'ARN comporte aussi 4 symboles $\mathcal{A}' = \{A, C, G, U\}$ où la thymine est remplacé par l'Uracil.

- Les protéines sont composées d'acides aminés qui sont au nombre de 20. L'alphabet sera:

$$\mathcal{A}'' = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$$

On pourra se rapporter aux tableaux de l'annexe A pour la correspondance entre les lettres et les acides aminés, et les relations entre l'ADN et les protéines.

En fait, il existe d'autres alphabets pour permettre la prise en compte d'informations incomplètes. Par exemple, si dans une séquence protéique, le séquençage n'a pas réussi à déterminer avec certitude l'acide aminé à une certaine place, on peut trouver la lettre $X$. De même, s'il reste une ambiguïté entre deux acides aminés, on peut trouver d'autres symboles. L'alphabet le plus courant est

$$\mathcal{B} = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V, B, Z, X, *\}$$

Une séquence protéique (resp. nucléique) est un mot construit avec l'alphabet des acides aminés (resp. des acides nucléiques). Une séquence $A$ s'écrit $A = A_1 A_2 \dots A_n$ où $A_i$ est le i-ème caractère de la séquence $A$. La longueur de la séquence $A$ est dans ce cas-là $n$.

Nous introduisons maintenant certaines notations relatives aux séquences. Soit A une séquence.

- $A_i$ ou $A[i]$ est la $i^{ème}$ lettre de la séquence $A$,
- $|A|$ représente la longueur de la séquence,
- $A_{i,j}$ ou $A[i, j]$ : représente la sous-séquence de $A$ entre les positions $i$ et $j$,
- $\epsilon$ est la chaîne de caractère vide,
- «.» représente l'opération de concaténation des séquences. Si $S_1 = ACGTGGTC$ et $S_2 = GTGCCA$, alors

$$S_1.S_2 = ACGTGGTCGTGCCA$$

### 1.1.2 Alignement de séquences

D'une manière informelle, l'alignement de deux séquences consiste à mettre en évidence les similitudes et les différences entre les deux séquences. On cherche à mettre en correspondance chacune des lettres de la première séquence avec l'une de la seconde. Evidemment, on s'offre aussi la possibilité de sauter quelques lettres.

Prenons un exemple. Soient les deux séquences $S_1 = ACGTCGTTC$ et $S_2 = AGGCCTCGC$. L'alignement entre ces deux séquences est:

```
S1 :   ACG--TCGTTC
       |  |   |||  |
S2 :   AGGCCTCG--C
```

Dans cet alignement, les paires d'acides nucléiques alignées sont mises face à face, alors que les lettres qui n'ont pas leur équivalent dans l'autre séquence sont en face d'un «−».

Une suite contigüe de «−» sera appelée un **décalage**, ou en anglais ***gap***, ou ***indel***[1], et correspond soit à une insertion soit à une délétion. Plus précisément, une **délétion** est un décalage dans la deuxième séquence (on passe de la première séquence à la seconde par la suppression de quelques lettres), alors qu'une **insertion** est un décalage dans la première séquence (on passe de la première séquence à la seconde en ajoutant quelques lettres). Cette différence entre insertion et délétion n'est que formelle, puisqu'elle dépend de l'ordre dans lequel on écrit l'alignement. Une insertion dans une séquence peut être vue comme une délétion associée à la seconde.

La mise en correspondance de deux lettres est appelée *appariement*. Deux cas peuvent se présenter :

- Les deux lettres sont identiques : on parlera alors d'*appariement exact.*[2]
- Les deux lettres diffèrent : on parlera d'*appariement non-exact.*[3]

Donnons un cadre général de l'alignement de séquences. Soit $\mathcal{A}' = \mathcal{A} \cup \{-\}$, l'alphabet sur lequel sera écrit l'alignement.

Un ***pseudo-alignement global*** $L$ entre $n$ séquences $\{S_1, S_2, \ldots S_n\}$ est un ensemble ordonné de mots $(\{S'_1, S'_2, \ldots S'_n\})$ construits sur l'alphabet $\mathcal{A}'$ ayant les contraintes suivantes :

- tous les mots ont la même longueur notée $|L|$,
- si on supprime dans chaque mot $S'_i$ les symboles «−», on obtient exactement la séquence $S_i$.

Un ***alignement global*** entre $n$ séquences $\{S_1, S_2, \ldots S_n\}$ est un pseudo-alignement global dont aucune des colonnes n'est vide : pour tout indice $k$ dans la séquence, il existe une séquence $S'_l$ dont le $k^{ième}$ caractère n'est pas le caractère «−» :

$$\forall k \in \{1, 2, \ldots |L|\} \quad \exists l \in \{1, 2, \ldots n\} \ \ tel \ \ que \ \ S'_l[k] \neq \text{«}-\text{»}$$

Un ***alignement local*** entre $n$ séquences $\{S_1, S_2, \ldots S_n\}$ est un alignement global sur des sous-séquences $\{S_1[l_1, r_1], S_2[l_2, r_2], \ldots S_n[l_n, r_n]\}$.

Dans le formalisme précédent, l'alignement de deux séquences n'est qu'un cas particulier de l'alignement multiple : le nombre de séquences est simplement égal à 2. Cependant, les méthodes d'alignement multiple ($n > 2$), sont très différentes de celles de l'alignement de deux séquences.

### 1.1.3 Alignement de deux séquences

Considérons deux séquences $A$ et $B$ de longueurs respectives $n$ et $m$. Nous pouvons définir l'alignement entre ces deux séquences en utilisant la notion de *pseudo-alignement*. Mais cette formulation n'est pas très constructive. Nous allons essayer de donner une autre définition de l'alignement de deux séquences. «*Aligner deux séquences, c'est mettre en correspondance des lettres de la première séquence avec des lettres de la seconde.*» Une paire de lettres ( l'une appartenant à la première séquence, et l'autre appartenant à la seconde ) sera appelée *paire alignée*.

1. **Indel** est la contraction de insertion/délétion.
2. On parlera de *match* en anglais.
3. On parlera de *mismatch* en anglais.

Formellement, si l'alphabet utilisé est $\mathcal{A}$, un alignement des séquences $A$ et $B$ est une suite de paires alignées: $\{(A_{i_k}, B_{j_k})_{1 \leq k \leq K}\}$ où les indices $i_k$ et $j_k$ vérifient:

- $i_k \leq i_{k+1}$ pour $1 \leq k \leq K-1$,
- $j_k \leq j_{k+1}$ pour $1 \leq k \leq K-1$,
- $i_k \in [1, n]$ et $j_k \in [1, m]$ pour $1 \leq k \leq K$.

Reprenons le formalisme précédent. Soit deux séquences $A$ et $B$ de longueur respective $n$ et $m$. soit $\mathcal{A}' = \mathcal{A} \cup \{-\}$, l'alphabet des alignements. Définissons une application de $\mathcal{A}'$ vers $\mathcal{A} \cup \{-\}$:

$$\begin{array}{rcccl} F & : & \mathcal{A}' & \longrightarrow & \mathcal{A} \cup \{-\} \\ & & a' & \longrightarrow & \left\{ \begin{array}{lll} a' & si & a' \neq - \\ \epsilon & si & a' = - \end{array} \right. \end{array} \tag{1.1}$$

Soit $\mathcal{P}(\mathcal{A})$ l'ensemble des paires possibles d'éléments de $\mathcal{A}' = \mathcal{A} \cup \{-\}$.

$$\mathcal{P}(\mathcal{A}) = \mathcal{A}'^2$$

Si $p \in \mathcal{P}(\mathcal{A})$, alors nous pouvons écrire $p$ sous la forme suivante: $p = \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix}$, où $a'_1 \in \mathcal{A}'$ et $a'_2 \in \mathcal{A}'$. Un alignement $L(A, B)$ est alors une suite d'éléments de $\mathcal{P}(\mathcal{A})$

$$L(A, B) = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix} \ldots \begin{bmatrix} a_{K'} \\ b_{K'} \end{bmatrix}$$

avec les contraintes suivantes:

$$\begin{bmatrix} a_i \\ b_i \end{bmatrix} \in \mathcal{P}(\mathcal{A}) \ \ pour \ 1 \leq i \leq K' \tag{1.2}$$

$$\begin{array}{c} F(a_1) \,.\, F(a_2) \,.\, F(a_3) \,.\, \ldots\ldots \,.\, F(a_{K'}) = A \\ F(b_1) \,.\, F(b_2) \,.\, F(b_3) \,.\, \ldots\ldots \,.\, F(b_{K'}) = B \end{array} \tag{1.3}$$

$K'$ est la longueur de l'alignement, mais le nombre de paires alignées K, est inférieur ou égal à $K'$.

A partir d'un alignement, on va essayer de définir un indice de similarité entre les deux séquences. On cherche à transformer l'une des deux séquences en l'autre. Pour cela, on dispose de trois opérations élémentaires sur les lettres:

- une opération de mutation: transformer une lettre en une autre (éventuellement la même),
- une opération de délétion: on supprime une lettre de la première séquence,
- une opération d'insertion: on rajoute une lettre à la deuxième séquence.

Les opérations de délétions et d'insertions sont complètement symétriques, et on parlera de **décalages** (*gap* ou *indel* en anglais) pour désigner soit une délétion soit une insertion.

On associe à chacune de ces opérations une valeur numérique que l'on appellera soit *score*, soit *coût*, soit *poids*. Lorsqu'on cherche un *indice de similarité* entre deux séquences, on calculera la somme des scores de ces opérations pour tous les alignements possibles et on en tirera le maximum. A l'inverse, pour un *indice de dissimilarité*, on minimisera la somme des coûts sur l'ensemble des

alignements possibles. Les deux approches sont équivalentes. On peut distinguer deux coûts de nature différente.

- Le score (ou coût) de la mutation de la lettre $A_i$ en $A_j$ dépend des 2 lettres $A_i$ et $A_j$ : plus les propriétés physico-chimiques des deux acides aminés (ou nucléiques) sont proches, plus le score (ou coût) sera important. Il pourra être négatif, si les deux acides sont franchement différents. L'ensemble des coûts de substitution est généralement donné par une matrice *symétrique*, noté $\sigma = (\sigma(a_i, b_j))_{\substack{1 \leq i \leq 20 \\ 1 \leq j \leq 20}}$.

  Dans le cas le plus simple, la matrice est la matrice identité. Si on reprend le formalisme de l'alignement, une paire alignée $p = \begin{bmatrix} a_i \\ b_j \end{bmatrix} \in \mathcal{P}(\mathcal{A})$ a un coût égal à $\sigma(a_i, b_j)$.

- les coûts des insertions/délétions sont fixés à une constante $g$. Ils ne dépendent pas de la position où ont lieu les insertions/délétions.

  Pour fixer les idées, on peut prendre comme coût de délétion $g = -1$. Pour $p = \begin{bmatrix} a'_i \\ b'_j \end{bmatrix} \in \mathcal{P}(\mathcal{A})$ avec $a'_i =$«$-$» ou (exclusif) $b'_j =$«$-$», le coût vaut $-1$.

On peut rassembler ces deux cas en donnant un coût $G$ à chaque paire de l'alignement :

$$\begin{array}{rcl} G \ : \ \mathcal{P}(\mathcal{A}) & \longrightarrow & \mathbb{R} \\ \begin{bmatrix} a'_i \\ b'_j \end{bmatrix} & \longrightarrow & \begin{cases} \sigma(a'_i, b'_j) & si \quad \begin{bmatrix} a'_i \\ b'_j \end{bmatrix} \ est\ une\ mutation \\ g & si \quad \begin{bmatrix} a'_i \\ b'_j \end{bmatrix} \ est\ une\ insertion/délétion \\ 0 & si \quad \begin{bmatrix} a'_i \\ b'_j \end{bmatrix} = \begin{bmatrix} - \\ - \end{bmatrix} \end{cases} \end{array} \tag{1.4}$$

L'alignement $L(A, B) = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix} \dots \begin{bmatrix} a_{K'} \\ b_{K'} \end{bmatrix}$ a un *score*

$$s\Big(L(A, B)\Big) = \sum_{k=1}^{k=K'} G\left(\begin{bmatrix} a'_i \\ b'_j \end{bmatrix}\right)$$

Le problème de l'alignement optimum est la recherche de l'alignement qui maximise le score de l'alignement $s(L(A, B))$.

$$\max_{L(A,B)} s\Big(L(A, B)\Big) = \max_{L(A,B)} \left[ \sum_{k=1}^{k=K'} G\left(\begin{bmatrix} a'_i \\ b'_j \end{bmatrix}\right) \right] \tag{1.5}$$

**Formulation du problème à l'aide d'un graphe orienté**

Ce problème peut se résoudre par les algorithmes de plus court chemin à travers un graphe orienté valué. Posons $\mathcal{G} = (E, V)$ le graphe orienté défini par :

- $E$ est l'ensemble des sommets du graphe : $E = \{(i, j) | i \in [0, n], j \in [0, m]\}$
- $V$ est l'ensemble des arcs :
  - si $i \in [1, n]$ et $j \in [1, m]$ alors il y a un arc partant du sommet $(i - 1, j - 1)$ vers le sommet $(i, j)$ représentant la substitution de $A_i$ en $B_j$ notée $\begin{bmatrix} A_i \\ B_j \end{bmatrix}$. A cet arc est associé un coût de $G(\begin{bmatrix} A_i \\ B_j \end{bmatrix})$. Ce sont les arcs obliques dans la figure 1.1.

- si $i \in [1, n]$ et $j \in [0, m]$ alors il y a un arc de $(i-1, j)$ vers $(i, j)$ représentant l'insertion de la lettre $B_j$ notée $\left[\begin{smallmatrix} - \\ B_j \end{smallmatrix}\right]$. Cet arc reçoit un coût de $G(\left[\begin{smallmatrix} - \\ B_j \end{smallmatrix}\right])$. Il s'agit des arcs verticaux de la figure 1.1.
- si $i \in [0, n]$ et $j \in [1, m]$ alors il y a un arc de $(i, j-1)$ vers $(i, j)$ représentant la suppression de la lettre $A_i$ notée $\left[\begin{smallmatrix} A_i \\ - \end{smallmatrix}\right]$. Cet arc reçoit un coût de $G(\left[\begin{smallmatrix} A_i \\ - \end{smallmatrix}\right])$. Ce sont les arcs horizontaux de la figure 1.1.

On spécifie ensuite l'unique source qui est le sommet $(0, 0)$ et l'unique puits $(n, m)$. Ainsi, le chemin de plus grand coût donnera l'alignement de plus grand score. La recherche du plus court chemin est un thème largement répandu en optimisation. Beaucoup d'algorithmes ont été proposés. En fait, dans le cas de l'alignement de séquences, les poids des arcs sont quelconques, mais le graphe a une structure très peu dense: d'un sommet ne partent que 3 arcs. La matrice des poids des arcs est alors tri-diagonale.

- La méthode générale est celle de Bellman, de type programmation dynamique résolvant le problème en $O(A \times S)$ où $A$ est le nombre d'arcs et $S$ est le nombre de sommets. Le nombre de sommets est dans notre cas $S = (n+1) \times (m+1)$, le nombre d'arcs étant de l'ordre de $A = 3mn$. L'algorithme de Bellman donne ici une complexité en $O(n^2m^2)$. D'Esopo et Pape ont proposé une version de cet algorithme ayant un très bon comportement moyen sur les graphes peu denses [104].

- Puisqu'il n'y a pas de circuit dans le graphe, l'algorithme de Bellman peut se simplifier grâce à une décomposition en niveaux [52]. Sa complexité tombe en $O(A)$, et devient dans le cas de l'alignement de séquences en $O(3mn)$. On est donc amené à calculer l'ensemble des chemins optimaux partant du sommet initial et allant à chacun des autres sommets.

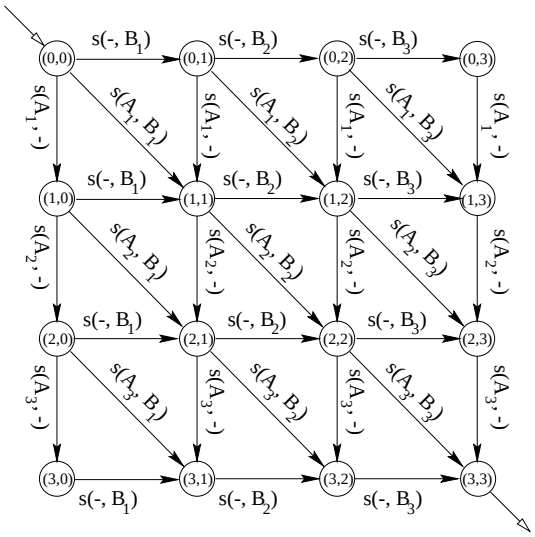



FIG. 1.1 - **La recherche de l'alignement global** *peut être vue comme la recherche du chemin optimal dans un graphe orienté valué.*

### 1.1.4 Matrices de substitution

Dans toutes les méthodes de comparaisons de séquences, apparaît une matrice de substitution. Cette matrice donne une notion de distance entre les acides nucléiques ou acides aminés. Le choix de la matrice a une grande influence sur les résultats de la méthode de comparaison. On ne trouvera pas les mêmes domaines de similitude si on compare plusieurs fois les deux mêmes séquences avec des matrices de substitution différentes.

D'autre part, chaque matrice de similarité repose sur un modèle différent de l'évolution. Comprendre le modèle sous-jacent permet de mieux choisir la matrice à utiliser pour résoudre tel ou tel problème.

Lorsqu'on aligne des séquences nucléiques, la matrice de substitution est souvent réduite à la matrice unité pour l'alignement local. Pour l'alignement global, d'autres matrices sont utilisées : on met une valeur positive sur la diagonale (par exemple +10), et une valeur négative à l'extérieur de cette diagonale ($-9$ par exemple).

Dans le cas de l'alignement des protéines, les mesures de similarité entre acides aminés sont plus complexes, et reposent sur certains modèles de l'évolution. Les matrices les plus utilisées sont nommées Dayhoff, PAM, Henikoff ou BLOSUM.

**Les matrices PAM :**

Le principe de la construction des matrices PAM n'est pas difficile. Nous essayons ici de formaliser les calculs permettant la construction de cette matrice. La première étape est de se donner un ensemble de séquences homogènes et de même longueur : $\mathcal{S} = \{S_i, 1 \leq i \leq K\}$ De plus on suppose que les alignements deux à deux sont connus. Nous noterons $S_k(l)$ la $l^{ième}$ lettre de la $k^{ième}$ séquence. $n_s(i)$ représentera le nombre de fois où la lettre $A_i$ apparaît dans la séquence $s$. On a bien sûr : $\sum_{1 \leq i \leq 20} n_s(i) = L_s$ où $L_s$ est la longueur de la séquence $s$. Posons $f_i$ la fréquence relative de la lettre $A_i$ sur l'ensemble des séquences de l'ensemble $\mathcal{S}$. Les fréquences relatives sont définies comme les fréquences d'apparition de chaque acide aminé sur l'ensemble des séquences considérées $\mathcal{S}$.

**Mutabilité relative non normalisée :** Posons $\mu_i(\{s, s'\})$ la mutabilité relative non normalisée d'un acide aminé $A_i$, pour une paire $\{s, s'\}$ de séquences alignées. C'est simplement le rapport entre le nombre de fois où l'acide aminé $A_i$ est présent dans l'une des deux séquences seulement, sur le nombre de fois où ce même acide aminé est présent dans l'une ou l'autre des 2 séquences. Autrement dit, il s'agit de la fréquence de mutation de la lettre $A_i$.

$$\mu_i(\{s, s'\}) = \frac{e_i(\{s, s'\})}{2.d_i(\{s, s'\}) + e_i(\{s, s'\})}$$

où $e_i(\{s, s'\})$ (esseulé) est le nombre de sites où on trouve, face à $A_i$, un autre acide aminé, et où $d_i(\{s, s'\})$ (doublet) est le nombre de sites où on trouve $A_i$ sur les 2 séquences. Cet indice n'est pas assez fin, puisqu'il est très sensible au pourcentage de mutation entre $s$ et $s'$. Introduisons le pourcentage de mutation $p(\{s, s'\})$, c'est-à-dire le nombre de sites où on ne trouve pas la même lettre sur les 2 séquences, divisé par la longueur commune des 2 séquences : $p(\{s, s'\}) = \frac{\sum_{i=1}^{20} e_i(\{s,s'\})}{2 \times L}$.

La mutabilité normalisée s'écrit alors :

$$m_i(\{s, s'\}) = \frac{\mu_i(\{s, s'\})}{p(\{s, s'\})}$$

Pour passer à un ensemble de séquences, il convient de totaliser les effets de chaque couple de séquences.

$$m_i = \frac{\sum e_i(\{s,s'\})}{\sum(2d_i(\{s,s'\}) + e_i(\{s,s'\})).p(\{s,s'\})}$$

D'autres définitions de la mutabilité relative à travers un ensemble de séquences [86] ont été données.
Remarques diverses:

- Le calcul de $m_i(\{s,s'\})$ ne nécessite pas que toutes les séquences soient de même longueur.
- S'il y a des délétions à l'intérieur d'un alignement, on peut tout de même envisager le calcul. Deux possibilités s'offrent à nous:
  - introduire une $21^{ème}$ lettre représentant la délétion,
  - ignorer les sites où il y a une insertion/délétion.
- Il peut y avoir plusieurs familles de séquences.

**Probabilité de mutation:** Notons $M_{\bar{ii}}$ la probabilité de mutation de la lettre $i$ vers une autre lettre, pour une période unitaire d'évolution. On pose que $M_{\bar{ii}}$ est proportionnelle à la mutabilité relative $m_i$.

$$M_{\bar{ii}} = \lambda.m_i$$

On a alors:

$$M_{ii} = 1.0 - \lambda.m_i$$

où $M_{ii}$ est la probabilité de persistance de l'acide aminé $i$. Posons

$$p_S = \sum_{1\leq i\leq 20} f_i.M_{ii}$$

la probabilité de rencontrer un acide aminé qui ne mute pas. On appelle $p_S$ la *probabilité de persistance à travers S*. On impose précisément à cette probabilité d'être égale à 0.99, ce qui correspond à une mutation pour 100 sites, pour une période unité d'évolution.

$$\begin{aligned} p_S &= \sum_{1\leq i\leq 20} p_S(i) = \sum_{1\leq i\leq 20} f_i.M_{ii} \\ &= \sum_{1\leq i\leq 20} f_i(1 - M_{\bar{ii}}) \\ &= 1 - \sum_{1\leq i\leq 20} f_i.M_{\bar{ii}} \end{aligned}$$

On en déduit la valeur de $\lambda$:

$$\begin{aligned} \sum_{1\leq i\leq 20} f_i.M_{\bar{ii}} &= \lambda.\sum_{1\leq i\leq 20} f_i.m_i = 0.01 \\ \lambda &= \frac{0.01}{\sum_{1\leq i\leq 20} f_i.m_i} \end{aligned} \tag{1.6}$$

On s'intéresse maintenant à la probabilité conditionnelle que $A_i$ mute en $A_j$ sachant que $i$ mute.

$$P^i_{ji} = A_{ji} / \sum_{k\neq i} A_{ki} \tag{1.7}$$

où $A_{ji}$ est le nombre de fois où on trouve face à l'acide aminé $A_i$ l'acide aminé $A_j$. On a alors :

$$\begin{aligned} M_{ji} &= M_{\bar{ii}} \times P^i_{ji} \\ &= \frac{\lambda.m_i.A_{ji}}{\sum_{k \neq i} A_{ki}} \end{aligned}$$

La matrice précédente $M_{ji}$ correspond à une période unitaire d'évolution, caractérisée par 1 % de mutation globale. Pour une période plus longue, de longueur $k$, on devra élever la matrice précédente à la puissance $k$. On obtiendra ainsi la matrice k-PAM (Point Accepted Mutation).

**La matrice de Dayhoff :**

La probabilité d'observer $A_j$ dans l'une des 2 séquences et $A_i$ dans l'autre est donnée par $(f_i.M_{ji} + f_j.M_{ij})$ dans le cas où $i \neq j$ et par $(f_i.Mii)$ si $i = j$. On pose :

$$R_{ji} = \frac{f_i.M_{ji} + f_j.M_{ij}}{2f_i.f_j} \;\; et \;\; R_{ii} = \frac{f_i.M_{ii}}{f_i^2} \tag{1.8}$$

La matrice $R$ est symétrique. On admet généralement que $f_i.M_{ji} = f_j.M_{ij}$, ce qui entraîne des simplifications pour le calcul de $R$.

$$R_{ji} = \frac{f_i.M_{ji}}{f_i.f_j} = \frac{M_{ji}}{f_j}$$

La matrice de Dayhoff est la matrice des logarithmes en base 10 des nombres $R_{ji}$ :

$$D = \{S_{ij} = Log_{10}(R_{ij}),\ 1 \leq i, j \leq 20\}$$

En fait la démarche de Dayhoff est la suivante. On compare pour deux séquences deux événements :

- les deux séquences sont indépendantes, donc la probabilité d'aligner l'acide aminé $i$ avec l'acide aminé $j$ est égale au produit des deux fréquences $f_i.f_j$.
- les deux séquences proviennent d'un ancêtre commun. Alors la probabilité que les deux acides aminés $i$ et $j$ soient alignés est :

$$\begin{aligned} P(i \;\; et \;\; j \;\; alignés) &= \sum_{x \in A} f_x M^t_{ix} M^t_{jx} \\ &= \sum_{x \in A} f_x M^t_{ix} M^t_{xj} \\ &= f_j M^{2t}_{ij} \end{aligned}$$

Les entrées de la matrice de Dayhoff sont les logarithmes des quotients de ces deux probabilités. La matrice $M^t$ est la matrice $M^{250}$.

L'algorithme de programmation dynamique (Cf. paragraphe 1.2) maximise la somme des similarités, donc la somme des logarithmes des quotients, donc le produit des quotients. L'algorithme trouve donc l'alignement qui maximise la probabilité que deux séquences proviennent d'un même ancêtre (maximum de vraisemblance).

**La matrice de Henikoff :**

On se donne tout d'abord un ensemble de séquences alignées. Posons $A_{ij}$ le nombre de sites dans l'ensemble des paires de séquences où les acides aminés $A_i$ et $A_j$ se retrouvent alignés.

On définit la *probabilité empirique* d'une rencontre entre $A_i$ et $A_j$ :

$$Q_{ij} = \frac{A_{ij}}{\sum_{1 \leq k,l \leq 20} A_{kl}}$$

Cette probabilité n'est pas une probabilité conditionnelle contrairement à celle de l'équation 1.7. On calcule ensuite

$$\begin{aligned} Q'_{ij} &= \frac{Q_{ij}}{2 f_i . f_j} \quad pour \quad i \neq j \\ Q'_{ii} &= \frac{Q_{ii}}{f_i . f_i} \end{aligned}$$

La matrice $Q'$ joue ensuite le même rôle que la matrice $R$ de l'équation 1.8, et la matrice de *Henikoff* sera :

$$H = \{2.log_2(Q'_{ij}), 1 \leq i, j \leq 20\}$$

Les matrices BLOSUM (*BLOcs SUbstitution Matrices*) reposent sur ce modèle, à partir d'alignements locaux. Les différentes matrices (Blosum45, Blosum62, Blosum80) ont été générées en changeant le niveau de similitude entre les séquences initiales. Blosum45 a été construite avec des séquences faiblement similaires.

Les différences entre la matrice Dayhoff et celle de Henikoff sont les suivantes :

- Dans le cas de Dayhoff on impose la probabilité de persistance à 0.99, alors que dans le cas de Henikoff, on constate cette probabilité de persistance (qui dépend de l'ensemble d'apprentissage).
- La notion de mutabilité relative permet de distribuer de façon proportionnelle la probabilité complémentaire de mutation. $M_{\bar{i}i}$ est réparti sur les différents acides aminés conformément à l'observation.

Un des inconvénients de ces méthodes est que l'alignement initial nécessite une matrice ! Dayhoff a utilisé des séquences très similaires, pour lesquelles l'alignement pouvaient être construit sans ambiguïté. Pour les matrices Blosum, une approche itérative a été utilisée : on aligne les séquences à partir d'une matrice arbitraire, on construit une nouvelle matrice, puis on itère ces deux phases jusqu'à ce que la matrice obtenue reste inchangée. Les matrices Blosum peuvent être obtenues à partir de différentes matrices initiales.

Les matrices PAM ou Blosum ne sont pas les seules, d'autres existent [86].

**Quelle matrice utiliser pour une recherche dans une base de données ?**

Après une étude fondée sur les principes de la théorie de l'information, Altschul [6] propose d'utiliser la matrice PAM120 pour la recherche des similarités locales sans insertion/délétion. Si l'on veut détecter des segments courts mais fortement homologues ou des alignements longs et faiblement homologues, on peut completer les résultats obtenus avec PAM120 par ceux obtenus avec PAM40 ou PAM250. Les autres matrices de la famille PAM n'apporteraint que peu d'information supplémentaire.

Cette remarque n'est pas valable pour les alignements avec insertion/délétion puisque on ne dispose d'aucune information sur la distribution des scores.

## 1.2 Programmation dynamique

Les algorithmes de Needleman & Wunsch et de Smith & Waterman permettent de calculer des scores et des alignements entre deux séquences biologiques A et B. Le premier donne le meilleur alignement *global* alors que le second donne le meilleur alignement *local*.

Dans toute cette section, on considère deux séquences A et B de longueur respectives $n$ et $m$ : $A = A_1 A_2 \dots A_n$ et $B = B_1 B_2 \dots B_m$.

### 1.2.1 Algorithme de Needleman & Wunsch

Needleman & Wunsch [101] ont été les premiers à utiliser la programmation dynamique pour comparer deux séquences biologiques. La programmation dynamique permet ici de construire un alignement global de score optimal entre les deux séquences $A$ et $B$.

Considérons l'exemple suivant. On cherche le meilleur alignement global entre la séquence $A = ATTACG$ et la séquence $B = ATATCG$. On associe un coût de 1 à une lettre alignée avec la même lettre, et un coût de 0 à toute lettre alignée avec une autre lettre. On attribue à toute insertion/délétion un coût de 0. Le score total est la somme des différents coûts pour chaque paire de l'alignement. Cette fonction de coût peut être représentée par une matrice de substitution de dimension $5 \times 5$ (Cf. tab. 1.1). Pour les protéines, on utilise systématiquement une matrice de substitution, dont les principes généraux de construction sont décrits en section 1.1.4.

| | - | A | C | G | T |
|---|---|---|---|---|---|
| - | | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 0 | 0 | 0 |
| C | 0 | 0 | 1 | 0 | 0 |
| G | 0 | 0 | 0 | 1 | 0 |
| T | 0 | 0 | 0 | 0 | 1 |

TAB. 1.1 - **Matrice de substitution pour l'ADN**

Le score d'un alignement est dans ce schéma de coût le nombre de lettres identiques alignées. L'alignement $\begin{pmatrix} A & T & T & A & C & G \\ A & T & A & T & C & G \end{pmatrix}$ a, dans ces conditions, un score de 4. Cependant, puisque les décalages sont permis, un score plus haut peut être atteint. Le score 5 est le maximum qu'on puisse atteindre. L'alignement optimal, c'est-à-dire un alignement pour lequel le score est maximum, est

$$\begin{matrix} A & T & - & T & A & C & G \\ A & T & A & T & - & C & G \end{matrix}$$

Cet alignement n'est pas le seul ayant un score optimal. On peut en donner un autre :

$$\begin{matrix} A & T & T & A & - & C & G \\ A & T & - & A & T & C & G \end{matrix}$$

L'algorithme de Needleman & Wunsch permet de réaliser un alignement global entre deux séquences de longueurs différentes. Cet alignement est obtenu par minimisation d'un coût appelé

*coût d'édition* :

$$M(i,j) = \max \begin{pmatrix} M(i-1,j-1) + S(A_i, B_j), \\ M(i-1,j) - \delta, \\ M(i,j-1) - \delta \end{pmatrix}$$

$M(i,j)$ correspond au score d'alignement des deux préfixes $A[1,i]$ et $B[1,j]$. En effet, si on cherche à maximiser ce score d'alignement, il faut prendre le maximum des scores d'alignement parmi l'ensemble des alignements possibles. On peut considérer trois situations *exclusives* pour la dernière paire d'alignement :

- Soit la dernière paire d'alignement correspond à un appariement, c'est-à-dire qu'elle est de la forme $\left[\begin{smallmatrix} A_i \\ B_j \end{smallmatrix}\right]$. Dans ce cas, le score d'alignement des préfixes $A[1,i]$ et $B[1,j]$ terminant par une mise en correspondance des lettres $A_i$ et $B_j$ s'écrit :

  $$S_1 = S(i-1,j-1) + G(\left[\begin{smallmatrix} A_i \\ B_j \end{smallmatrix}\right])$$

  où $S(i-1,j-1)$ correspond au meilleur score pour aligner les sous-séquences $A[1,i-1]$ et $B[1,j-1]$.

- Soit elle correspond à une insertion sur la séquence $A$. Dans ce cas, elle est de la forme $\left[\begin{smallmatrix} A_i \\ - \end{smallmatrix}\right]$. Ceci implique que les sous-séquences $A[1,i-1]$ et $B[1,j]$ ont été alignées. Le meilleur score d'alignement des préfixes $A[1,i]$ et $B[1,j]$ terminant par une insertion de la lettre $A_i$, s'exprime par :

  $$S_2 = S(i-1,j) + G(\left[\begin{smallmatrix} A_i \\ - \end{smallmatrix}\right]).$$

- Soit elle correspond à une délétion sur la séquence $A$, et elle s'exprime par $\left[\begin{smallmatrix} - \\ B_i \end{smallmatrix}\right]$. De la même manière, le score d'alignement maximum entre les préfixes $A[1,i]$ et $B[1,j]$ terminant par une insertion de la lettre $B_j$ s'exprime par :

  $$S_3 = S(i,j-1) + G(\left[\begin{smallmatrix} - \\ B_j \end{smallmatrix}\right]).$$

Le maximum des trois scores $S_1$, $S_2$ et $S_3$ correspond au score maximum pour aligner les deux séquences $A[1,i]$ et $B[1,j]$, quelle que soit la dernière paire d'alignement.

L'algorithme de Needleman & Wunsch (NW) recherche pour chaque paire de préfixes $A[1,i]$ et $B[1,j]$ quelle est la situation parmi les trois possibles (insertion, délétion ou appariement) qui mène au score optimal. Dans cette version de la programmation dynamique, seul le score est calculé et l'alignement n'est pas construit.

La première méthode pour calculer l'alignement est de parcourir la matrice entière en sens contraire. Lors de la première phase, on a parcouru les séquences dans le sens de lecture. Lors de la phase de remonté, on peut, pour chaque case, retrouver la case qui a permis d'obtenir le meilleur score (Cf. figure 1.2). L'inconvénient de cette **remontée** est la nécessité, lors de la phase de **descente**, de stocker, pour chaque couple $(i,j)$, le chemin emprunté pour obtenir le score $M_{i,j}$. La complexité en espace de cet algorithme est alors $O(m \times n)$ où $m$ et $n$ sont les longueurs respectives des deux séquences.

Une première amélioration a été donnée par Waterman, Smith et Beyer [130], qui ont introduit un coût pour les insertions/délétions dépendant de la longueur de ce décalage. En effet, biologiquement l'événement le plus rare est bien l'apparition d'une insertion/délétion. L'événement correspondant à l'allongement d'une insertion/délétion est nettement plus répandu. Ceci implique que la

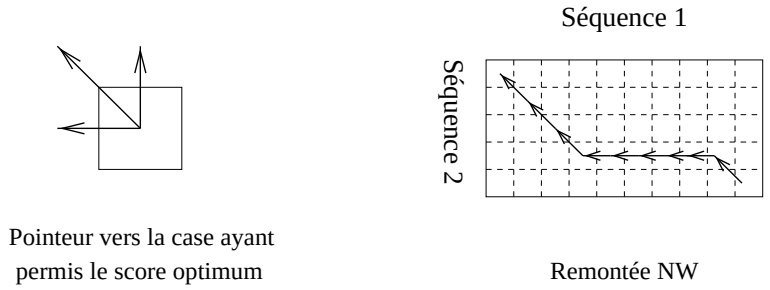



FIG. 1.2 - **Principe de la remontée pour l'algorithme Needleman & Wunsch :** *Grâce aux pointeurs stockés lors de la phase de descente, l'alignement peut être exhibé très rapidement.*

fonction de pénalisation d'une insertion/délétion de longueur $k$ notée $g(k)$ doit être une fonction concave et croissante de la longueur de l'insertion/délétion.

On peut réécrire l'algorithme de NW en prenant en compte cette fonction de pénalisation des insertions/délétions :

$$M(i,j) = \max \left( \begin{array}{c} M(i-1,j-1) + S(A_i, B_j), \\ \max_{1 \leq k \leq i}\{M(i-k,j) - g(k)\}, \\ \max_{1 \leq k \leq j}\{M(i,j-k) - g(k)\} \end{array} \right) \tag{1.9}$$

Biologiquement, l'opération qui coûte cher est la création d'une insertion/délétion. Plus elle est longue, plus le surcoût dû à son allongement est faible. Pour rendre compte de ce modèle biologique, la fonction de pénalisation des insertions/délétions doit être concave.

Dans cette expression, on peut se rendre compte que la complexité a beaucoup augmenté par rapport à celle de l'algorithme initial de NW. Dans le cas de l'algorithme de base, la complexité en temps est de l'ordre de $O(n \times m)$, puisqu'on parcourt toutes les cases d'un tableaux de dimension $n \times m$. Pour chaque case, la complexité du calcul est toujours la même : 3 additions, et 2 maximisations. Ici, la complexité pour chaque case du tableau croît avec les indices du tableau, puisqu'elle est en $O(i+j)$. Nous reviendrons plus tard sur les développements dus à Miller et Myers [92].

Il existe cependant un cas où la complexité en temps n'est pas beaucoup augmentée, il s'agit du cas où la fonction de pondération des insertions/délétions s'exprime de manière affine :

$$g(k) = go + ge \times (k-1)$$

Pour respecter la concavité de la fonction de pénalisation, on a $go \geq ge$. Les paramètres $go$ et $ge$ sont généralement appelés respectivement *gap-open penalty* [4] et *gap-extend penalty* [5] . Dans ce cas là, il n'est pas nécessaire de considérer le maximum $\max_{1 \leq k \leq i}\{M(i-k,j) - g(k)\}$. Supposons que

4. noté par la suite *gapo* ou *go*.
5. noté par la suite *gape* ou *ge*.

l'on connaisse le score du meilleur alignement terminant par une insertion en $(i-1,j)$.

$$\begin{aligned} S_2(i,j) &= \max_{1\le k\le i}\Big(M(i-k,j)-g(k)\Big) \\ &= \max\Big(\max_{2\le k\le i}\{M(i-k,j)-g(k)\}, M(i-1,j)-g(1)\Big) \\ &= \max\Big(\max_{2\le k\le i}\{M(i-k,j)-g(k-1)\}-ge, M(i-1,j)-g(1)\Big) \\ &= \max\Big(S_2(i-1,j)-ge, M(i-1,j)-g(1)\Big) \end{aligned}$$

Le même calcul s'applique au cas de la délétion. Le calcul pour chaque couple d'indice $(i,j)$ nécessite en plus 2 additions et 2 opérations de maximisation. L'algorithme reste en $O(n\times m)$.

On présente généralement cet algorithme avec trois tableaux:

- $N(i,j)$ représentant le score d'alignement des préfixes $A[1,i]$ et $B[1,j]$,
- $D(i,j)$ représentant le score du meilleur alignement se terminant par une délétion, et
- $I(i,j)$ représentant le score du meilleur alignement se terminant par une insertion.

On a alors la récurrence suivante:

$$N(i,j) = \max\Big(\ N(i-1,j-1)+S(A_i,B_j),\ D(i,j),\ I(i,j)\ \Big) \tag{1.10}$$

$$I(i,j) = \max\begin{pmatrix} N(i,j-1)-go, \\ I(i,j-1)-ge \end{pmatrix} \tag{1.11}$$

$$D(i,j) = \max\begin{pmatrix} N(i-1,j)-go, \\ D(i-1,j)-ge \end{pmatrix} \tag{1.12}$$

Il est évident que les initialisations pour cet algorithme doivent être homogènes avec la boucle de l'algorithme. Elles sont données par:

$$\begin{array}{lllll} N(0,0) = I(0,0) = D(0,0) = 0 & & \\ D(i,0) = I(i,0) = -g(i) & & 1\le i\le n \\ D(0,j) = I(0,j) = -g(j) & & 1\le j\le m \end{array}$$

La notion d'alignement global est à opposer à celle d'alignement local, pour lequel on recherche les parties de séquences les plus semblables.

### 1.2.2 Algorithme de Smith & Waterman

La détermination des sous-séquences homologues parmi un ensemble de longues séquences est aussi importante pour l'analyse de séquences biologiques. Le problème est plus simple si on ne considère que des alignements qui ne contiennent pas d'insertion/délétion. Dans ce cas, l'algorithme BLAST [5] permet de sélectionner toutes les séquences d'une banque qui possèdent une sous-séquence dont l'alignement sans insertion/délétion avec une sous-séquence de la séquence *query*, a un score supérieur à un seuil donné (Cf. section 1.3.1).

Si on se pose le problème similaire en permettant les insertions/délétions, une solution efficace a été donnée par Smith & Waterman. Ils ont repris et modifié l'algorithme de Needleman & Wunsch.

Smith & Waterman définissent une autre variable score, appelée score SW, qui est définie comme le score optimal entre deux sous-séquences quelconques. Il s'agit d'une optimisation du score de Needleman & Wunsch sur l'ensemble des sous-séquences possibles [116]. Pour deux séquences $A$ et $B$ de longueurs $n$ et $m$, on a :

$$W(A[1,n], B[1,m]) = \max_{\substack{I,J \\ N(I,J)\geq 0}} N(I,J)$$

où $I$ et $J$ sont deux segments de $[1,n]$ et $[1,m]$ respectivement (Cf. figure 1.3). Dans le cas de l'algorithme Smith & Waterman, on ne retiendra que les segments $I$ et $J$ tels que $N(I,J) \geq 0$. Ces segments existent dès lors que l'on peut trouver deux lettres $A_i$ et $B_j$ telles que $S(A_i, B_j) \geq 0$.

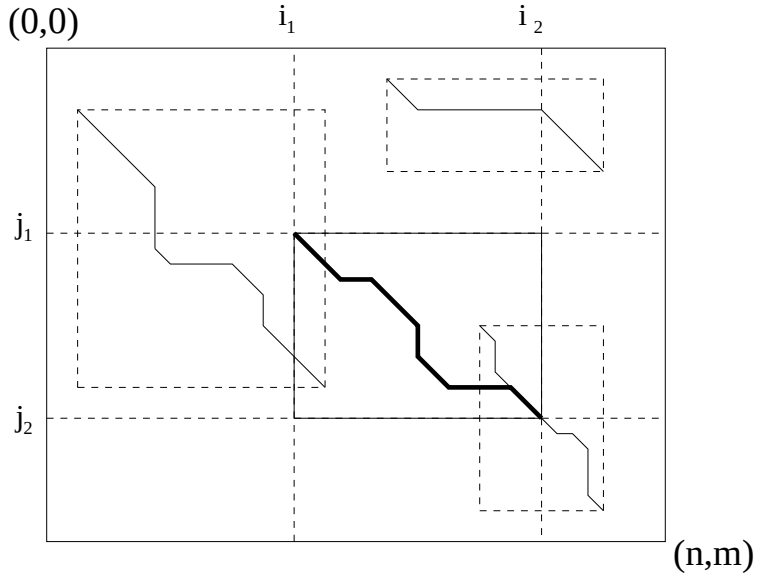



FIG. 1.3 - **L'algorithme Smith & Waterman** *vu comme une optimisation du score Needleman & Wunsch sur l'ensemble des sous-séquences possibles. On cherche les sous-séquences définies par les indices* $i_1 \in [1,n]$, $i_2 \in [i_1+1, n]$, $j_1 \in [1,m]$, $j_2 \in [j_1+1, m]$ *qui maximisent le score* **Needleman & Wunsch**.

Pour répondre à ce problème nous introduisons une nouvelle variable $W'(i,j)$ dont l'algorithme a été donné par Smith & Waterman. Nous montrerons ensuite que le calcul de la valeur maximale de $W'(i,j)$, sur l'ensemble des indices, appelée score Smith & Waterman, équivaut au calcul de la variable $W(A,B)$.

Notons $W'(i,j)$ le meilleur score Needleman & Wunsch parmi toutes les sous-séquences de $A$ et $B$ de la forme $A[i_1,i]$ et $B[j_1,j]$. On obtient la relation de récurrence suivante :

$$W'(i,j) = \max \begin{pmatrix} W'(i-1,j-1) + S(A_i, B_j), \\ W'(i-1,j) - \delta, \\ W'(i,j-1) - \delta, \\ 0 \end{pmatrix} \tag{1.13}$$

En effet, quatre possibilités s'offrent à nous, pour terminer l'alignement en $A_i$ et $B_j$ :

- Soit $A_i$ et $B_j$ sont en correspondance, alors le meilleur score entre deux sous-séquences par un appariement des lettres $A_i$ et $B_j$ est :

$$W'(i-1, j-1) + S(A_i, B_j)$$

- Soit l'alignement finit par une délétion, on a alors dans l'alignement la paire $\begin{bmatrix} A_i \\ - \end{bmatrix}$, et le score devient :

$$W'(i-1, j) - \delta$$

- Soit l'alignement finit par une une insertion dans la séquence $B$, on a alors dans l'alignement la paire $\begin{bmatrix} - \\ B_j \end{bmatrix}$, et le score devient :

$$W'(i, j-1) - \delta$$

- Soit il n'existe pas de sous-séquences terminant par $A_i$ et $B_j$ ayant une similarité positive. Alors le score est simplement 0.

Les initialisations du tableau sont différentes de celles pour l'algorithme Needleman & Wunsch :

$$\begin{array}{lcll} W'(0,0) & = & 0 & \\ W'(i,0) & = & 0 & 1 \leq i \leq n \\ W'(0,j) & = & 0 & 1 \leq j \leq m \end{array}$$

Ces initialisations sont cohérentes avec le résultat que l'on souhaite avoir : les insertions/délétions de début d'alignement doivent toutes avoir un score nul. Dès l'initialisation, le calcul de $W'$ ne prend pas en compte les scores négatifs dus aux insertions/délétions de début d'alignement.

On peut tout à fait reprendre le cas général de l'algorithme de Needleman & Wunsch qui prend en compte des pénalités différentes suivant la taille des insertions/délétions. Ici, on ne doit prendre en compte que les zones de similarités positives : lorsqu'on obtient un score inférieur à 0, on le remplace par 0. Soit $g(k)$ la fonction de pénalisation des insertions/délétions. L'algorithme s'écrit :

$$W'(i,j) = \max \begin{pmatrix} W'(i-1, j-1) + S(A_i, B_j), \\ \max_{1 \leq k \leq i}\{W'(i-k, j) - g(k)\}, \\ \max_{1 \leq k \leq j}\{W'(i, j-k) - g(k)\} \\ 0 \end{pmatrix}$$

Dans le cas où la fonction de pondération des insertions/délétions s'exprime de manière affine : $g(k) = go + ge \times (k-1)$, on introduit les trois tableaux précédemment définis, auxquels on donne la même signification en remplaçant le score NW par le meilleur score SW parmi toutes les sous-séquences terminant par $A_i$ et $B_j$. On a la récurrence suivante :

$$W'(i,j) = \max \begin{pmatrix} W'(i-1, j-1) + S(A_i, B_j),\ I(i,j),\ D(i,j),\ 0 \end{pmatrix} \quad (1.14)$$

$$I(i,j) = \max \begin{pmatrix} W'(i, j-1) - go, \\ I(i, j-1) - ge \end{pmatrix} \quad (1.15)$$

$$D(i,j) = \max \begin{pmatrix} W'(i-1, j) - go, \\ D(i-1, j) - ge \end{pmatrix} \quad (1.16)$$

Les initialisations pour cet algorithme sont :

$$\begin{array}{lllllll} W'(0,0) & = & I(0,0) & = & D(0,0) & = & 0 \\ D(i,0) & = & I(i,0) & = & 0 & & 1 \leq i \leq n\,, \\ D(0,j) & = & I(0,j) & = & 0 & & 1 \leq j \leq m\,. \end{array}$$

De la même manière que pour l'algorithme Needleman & Wunsch, l'algorithme présenté ci-dessus ne donne pas l'alignement. Pour l'obtenir, la méthode la plus simple est de parcourir la matrice entière en sens contraire. Lors de la première phase, on a parcouru les séquences dans le sens de lecture. Une fois qu'on a trouvé la position de la fin de l'alignement, on peut, pour chaque case, retrouver la case qui a permis d'obtenir le meilleur score (Cf. figure 1.4). Lors de cette remontée, on s'arrête quand le score obtenu sur le chemin de la remontée est égal au score trouvé lors de la phase de descente. L'inconvénient est encore le même : on est obligé, lors de la phase de descente, de stocker, pour chaque couple $(i, j)$, le chemin emprunté pour obtenir le score $M_{i,j}$. La complexité en espace de cet algorithme est alors $O(m \times n)$ où $m$ et $n$ sont les longueurs respectives des deux séquences. Grâce à cette remontée, on peut avoir l'alignement complet, en particulier les

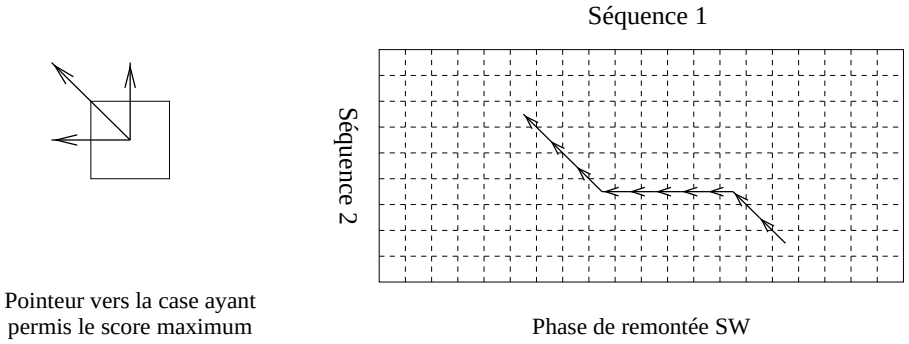



FIG. 1.4 - **Principe de la remontée pour l'algorithme Smith & Waterman :** *Grâce aux pointeurs stockés lors de la phase de descente, l'alignement peut être exhibé très rapidement.*

positions de début et fin d'alignement.

**Remarques :** Quelques remarques peuvent être faites sur l'alignement optimal SW. Plaçons-nous dans le cas de la figure 1.3, où l'alignement optimal se trouve entre les sous-séquences $A[i_1 + 1, i_2]$ et $B[j_1 + 1, j_2]$.

- Le chemin SW commence par une substitution positive entre $A[i_1+1]$ et $B[j_1+1]$, et termine aussi par une substitution positive entre $A[i_2]$ et $B[j_2]$. Autrement dit, lorsque l'alignement comporte au moins deux substitutions, il y en a au moins deux de coûts positifs : $(A[i_1 + 1], B[j_1 + 1])$ et $(A[i_2], B[j_2])$ (Cf. figure 2.13, p. 73).

  Examinons ce qui se passe en $(i_2, j_2)$. Si le chemin se terminait par une insertion/délétion, il existerait un $k$ tel que $W'(i_2 - k, j_2)$ ou $W'(i_2, j_2 - k)$ serait supérieur à $W'(i_2, j_2)$. Ceci n'est pas possible puisque $W'(i_2, j_2)$ est la valeur maximale des $W'(i, j)$ sur l'ensemble du tableau. Si le chemin se terminait par une substitution strictement négative, alors $W'(i_2 - 1, j_2 - 1) > W'(i_2, j_2)$.

  Le raisonnement est le même pour le début de l'alignement.

- Si on s'intéresse au sous-tableau $[i_1, i_2] \times [j_1, j_2]$, le chemin pris par l'algorithme SW entre $(i_1, j_1)$ et $(i_2, j_2)$ est le même que le chemin pris par NW sur les séquences $A[i_1 + 1, i_2]$ et $B[j_1 + 1, j_2]$, ou du moins est un chemin optimal pour NW sur ces deux sous-séquences.
- Si on prend les sous-séquences $A[i_1 + 1, i_2]$ et $B[j_1 + 1, j_2]$, toutes les valeurs $NW(i, j)$ (sur ces sous-séquences) sont inférieures ou égales à celles $SW(i, j)$. Elles sont égales pour tout couple d'indices $(i, j)$ tel que $SW(i, j) > 0$, $(i, j)$ appartenant à l'alignement optimal.

La section 2.5 détaille ces remarques, qui permettent de montrer que l'algorithme de Smith & Waterman répond bien à la question qu'on se pose : trouver $I$ et $J$ tels que $W(A, B) = \max_{I,J} N(I, J)$ où $I$ et $J$ sont deux segments de $[1, n]$ et $[1, m]$ respectivement. On a donc :

$$\max_{i,j} W'(i, j) = W(A, B) = \max_{I,J} N(I, J).$$

Par abus de notation, on notera $W(i, j) = W'(i, j)$ le score obtenu par l'algorithme SW pour les indices $i$ et $j$. N'oublions pas cependant qu'il peut y avoir plusieurs alignements locaux de score maximal :

$$\begin{aligned} W(I_1, J_1) &= \max_{I,J} N(I, J) \\ W(I_2, J_2) &= \max_{I,J} N(I, J) \\ W(I_3, J_3) &= \max_{I,J} N(I, J) \end{aligned}$$

et qu'il peut exister plusieurs chemins optimaux terminant en $(i_2, j_2)$ (Cf. figure 1.5). Nous reviendrons plus tard sur ce problème (Cf. section 2.7).

**Remarque :** Les scores SW dépendent des matrices de substitutions. Le score final est la somme, sur le chemin optimal, des coûts des substitutions à laquelle on a retranché la somme des pénalisations des insertions/délétions.

Lorsqu'on rajoute une constante à tous les éléments d'une matrice de substitution, l'alignement SW se trouve modifié. Plus il y a de valeurs positives dans la matrice, plus l'alignement sera long. En effet, un alignement est coupé lorsqu'il n'est plus possible de l'allonger tout en augmentant le score. Si tous les éléments de la matrice sont positifs, une substitution supplémentaire ne pourra qu'augmenter le score, et l'alignement s'allongera. Il s'agit d'un moyen pour jouer sur la longueur de l'alignement.

### 1.2.3 La programmation dynamique vue comme un problème de recherche de chemin optimal dans un graphe valué

#### Alignement global

L'algorithme de NW répond parfaitement au problème initial (éq 1.5) de l'alignement global. On peut trouver un graphe, pour lequel la recherche du chemin le plus rentable est équivalent à la recherche de l'alignement optimal. Puisque dans la formulation adoptée, on cherche un alignement de similarité maximale, le chemin optimal pour ce graphe sera le chemin qui maximise la somme des coûts élémentaires de chacun de ces arcs (Cf. figure 1.6).

Lorsque la fonction de pénalisation d'une insertion/délétion est linéaire, le graphe associé a exactement $(n + 1) \times (m + 1)$ sommets (Cf. figure 1.1). Lorsqu'on passe à une fonction affine, le graphe a trois fois plus de sommets (Cf. figure 1.6). Ceci est dû au fait que la pénalisation d'une

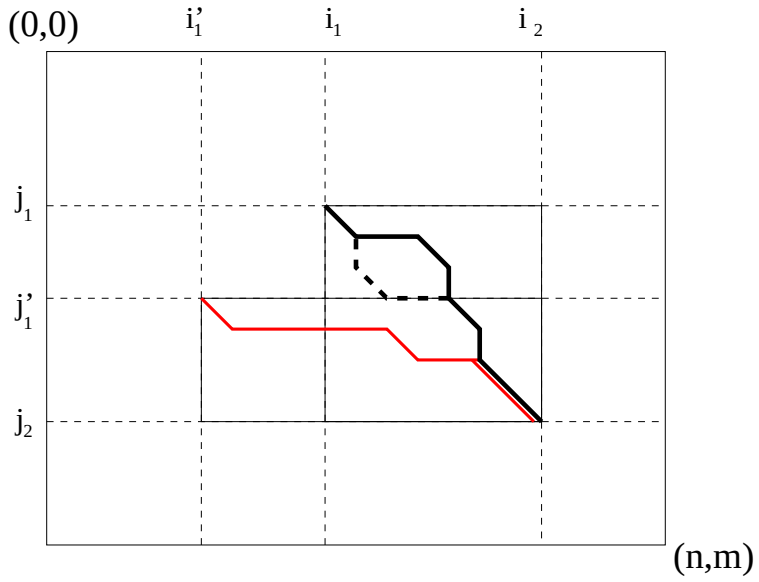



FIG. 1.5 - **Score Smith & Waterman :** *Un score optimal correspondant à un alignement se terminant en $(i_2, j_2)$ peut être obtenu par différents chemins. Deux chemins différents peuvent ou non provenir d'une même origine. Ici, deux chemins optimaux existent entre les sous-séquences $A[i_1, i_2]$ et $B[j_1, j_2]$, ces deux chemins donnent deux alignements différents entre les deux mêmes sous-séquences. Un troisième alignement optimal existe entre les sous-séquences $A[i'_1, i_2]$ et $B[j'_1, j_2]$.*

extension d'une insertion/délétion n'est pas égale à celle de la création de l'insertion/délétion. On a 6 types d'arcs :

- les arcs «*diagonaux*» modélisant les appariements,
- les arcs du type $D(i,j) \longrightarrow V(i+1,j)$ modélisant la création d'une insertion, le coût associé à cet arc est *gapo*,
- les arcs du type $V(i,j) \longrightarrow V(i+1,j)$ modélisant l'extension d'une insertion, le coût associé à cet arc est *gape*,
- les arcs du type $D(i,j) \longrightarrow H(i,j+1)$ modélisant la création d'une délétion, le coût associé à cet arc est *gapo*,
- les arcs du type $H(i,j) \longrightarrow H(i,j+1)$ modélisant l'extension d'une délétion, le coût associé à cet arc est *gape*,
- les arcs du type $H(i,j) \longrightarrow D(i,j)$, ou $V(i,j) \longrightarrow D(i,j)$ modélisant la fin d'une insertion/délétion, le coût associé à cet arc est nul.

  Ce n'est pas la seule manière de modéliser la fin d'une insertion/délétion. On peut par exemple rajouter les arcs $H(i,j) \longrightarrow D(i,j+1)$ et $V(i,j) \longrightarrow D(i+1,j)$. Cependant, la représentation choisie a l'avantage de pouvoir intégrer une pénalisation de fin d'insertion/délétion.

On peut alors répartir la pénalité d'ouverture d'une insertion/délétion sur l'ouverture et la fermeture des trous.

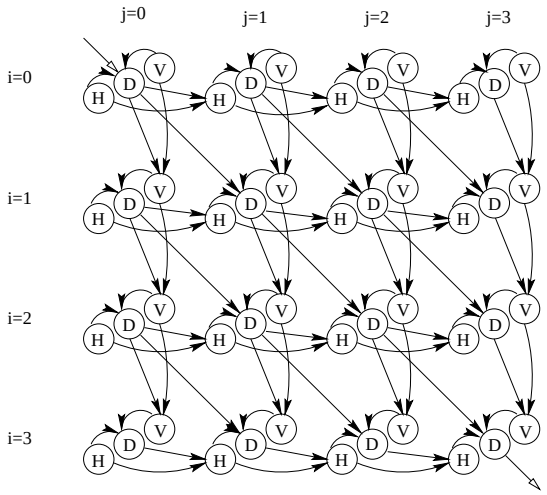



FIG. 1.6 - **Graphe associé à l'algorithme de l'alignement global** *dans le cas d'une fonction affine de pénalisation des insertions/délétions. Cf. figure 1.1 pour le cas linéaire.*

L'algorithme de Needleman & Wunsch est strictement équivalent à la recherche du plus court chemin dans un graphe valué *sans circuit*. La complexité de l'algorithme NW est la même que celle des meilleurs algorithmes de recherche de plus court chemin dans un graphe valué *sans circuit*. Dans le cas général, la complexité de l'algorithme de Bellman pour la recherche du plus court chemin est en $O(S \times A)$, où $S$ et $A$ sont respectivement le nombre de sommets et le nombre d'arcs du graphe. S'il n'y a pas de circuit, l'algorithme passe en $O(S)$. On pourra se référer pour ce type de problèmes à [52].

### Alignement local

Le problème de l'alignement local est un problème très similaire. Mais, le graphe est légèrement différent en ce qui concerne l'ensemble des sommets initiaux et finaux. Lorsque la fonction de pénalisation des insertions/délétions est linéaire, la graphe associé à l'algorithme NW n'a qu'un seul sommet initial et qu'un seul sommet final (Cf. figure 1.1). Dans le cas de SW, tous les sommets sont à la fois initiaux et finaux.

Lorsque la fonction de pénalisation des insertions/délétions est affine, seuls les sommets correspondant à des appariements doivent être considerés comme initiaux et finaux, puisque des autres sommets ne partent que des arcs de coût négatif.

La théorie des graphes appelle *un distancier* l'ensemble des chemins de poids minimum entre deux sommets quelconques d'un graphe valué. Dans le cas de l'algorithme de Smith & Waterman, on ne détermine que *l'unique* score optimal, et éventuellement le chemin associé. Cet algorithme n'a pas d'équivalent pour la théorie des graphes, bien qu'il soit tout à fait applicable au problème de la recherche *du* chemin de coût minimum (quels que soient les sommets de début et de fin) dans un graphe valué *sans circuit*.

### 1.2.4 Complexité de l'algorithme Smith & Waterman

Les premières implémentations de la programmation dynamique effectuaient tous les calculs sans tenir compte de quelques propriétés simples. L'algorithme est alors stable, c'est-à-dire que le temps d'exécution ne dépend pas des séquences que l'on compare. Sa complexité est en $O(n \times m)$ où $n$ et $m$ sont les longueurs respectives des deux séquences à comparer. Il a fallu attendre 1992, pour que les propriétés de l'algorithme soient exploitées par Phil Green [54] :

- Si $W(i,j) < g(1)$ alors les opérations max pour le calcul de $I(i+1,j)$ et de $D(i,j+1)$ sont inutiles. Si $W(i,j) < g(1)$, une insertion/délétion rend le score négatif, et la valeur 0 est choisie d'office.
- Les additions concernant $W(i,j)$ dans $I(i+1,j) = \max\begin{pmatrix} W(i,j) - go, \\ I(i,j) - ge \end{pmatrix}$ et $D(i,j+1) = \max\begin{pmatrix} W(i,j) - go, \\ D(i,j) - ge \end{pmatrix}$ peuvent être effectuées en même temps.
- Si $I(i,j) < 0$ alors on peut ne pas effectuer l'opération max pour le calcul de $I(i,j+1)$ et une des deux opérations max pour le calcul de $N(i,j+1)$.
- Enfin, puisqu'on s'intéresse au meilleur score, on peut considérer que les scores inférieurs à $g(1)$ n'ont pas à être reportés. Cette supposition réaliste permet de diminuer le nombre de mises à jour du score final. Si on prend un $gapo = 0$, l'algorithme est légèrement plus long que l'implémentation habituelle.

La complexité au pire est légèrement supérieure à celle de l'implémentation habituelle, mais la complexité en moyenne est plus faible. En pratique, sur des banques de données réelles, l'implémentation de Green est 25% à 30% plus rapide que l'implémentation habituelle.

| | | en moyenne | au pire |
|---|---|---|---|
| implémentation habituelle | max | $6.n.m$ | |
| | + | $5.n.m$ | |
| implémentation de Green | max | $5.m.n$ | $7.n.m$ |
| | + | $2.666\,m.n$ | $4.n.m$ |

TAB. 1.2 - **Complexité de l'algorithme de Smith & Waterman**

## 1.3 Les heuristiques les plus utilisées

### 1.3.1 $1^{ère}$ heuristique : BLAST

BLAST ([5]) est une méthode heuristique pour trouver les meilleurs alignements locaux de plus grands scores, entre une séquence donnée appelée *séquence requête* et une banque de séquences. BLAST est l'acronyme de *Basic Local Alignment Search Tool*. Il est important de noter que BLAST ne permet pas d'insertion/délétion, mais l'algorithme permet de trouver plusieurs régions similaires (*matches*) pour une même séquence de la banque. L'algorithme de BLAST (ainsi que ses variantes), est fondé sur les travaux de Karlin et Altschul sur les alignements de séquences sans insertion/délétion ([78]). Ainsi, on peut donner la probabilité d'obtenir un alignement de score

donné. L'algorithme de BLAST permet de localiser presque toutes les régions similaires, souvent appelées MSP (Maximal Segment Pair) dont le score dépasse une valeur seuil, et ceci de manière efficace.

L'algorithme se décompose en quatre phases :

1. Pour une longueur de mot donnée $w$, généralement 3 pour les protéines, et pour une matrice de substitution, on crée une liste de tous les mots de longueur $w$, qui peuvent avoir un score supérieur ou égal à $T$, lorsqu'ils sont comparés aux mots de longueurs $w$ de la *séquence requête.*

2. On cherche dans la banque de séquences, toutes les occurrences de ces mots.

3. Pour chaque localisation de ces mots de longueur fixe, dans la banque, on cherche à étendre l'alignement de part et d'autre pour obtenir un alignement de score $\geq S$. $S$ est le seuil nécessaire pour retenir un MSP (Maximal Segment Pair). Puisque la matrice de coût contient des valeurs négatives, l'extension d'un alignement, peut faire décroître le score. Dès que le score passe en dessous d'une limite $X$, on termine l'extension. En règle générale, on a $X < S$, pour permettre de raccrocher deux zones similaires dans le même MSP. En fait, il y a un va-et-vient entre l'extension à droite et l'extension à gauche : il est possible que le gain obtenu par l'extension de l'un des côtés puisse permettre l'extension de l'autre, même si l'extension précédente avait été stoppée.

4. L'algorithme élimine les alignements les moins significatifs. Puisqu'il y a un modèle probabiliste sous-jacent, on supprime tous les scores dont l'espérance est plus grande qu'un certain seuil.

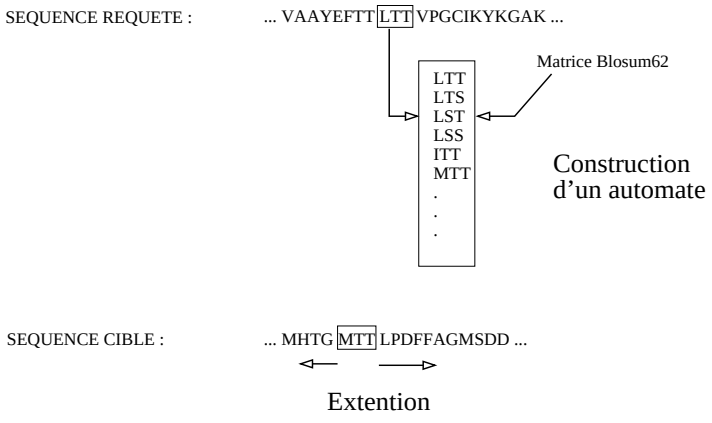



FIG. 1.7 - **Les étapes de l'algorithme BLAST.**

Une valeur faible de $T$ réduit la possibilité de manquer un alignement de score $S$ (augmentation de la sensibilité). Mais la diminution de la valeur de $T$ entraîne une augmentation de la liste des mots générés dans la phase 2. Le temps d'exécution et l'espace mémoire nécessaire augmentent de manière significative.

Malheureusement, BLAST n'est pas aussi sensible que la programmation dynamique. Cependant les statistiques sous-jacentes donnent une estimation directe de la signification d'un alignement. Le programme a été développé au NCBI[6], et bénéficie de supports techniques importants. Des filtres ont récemment été développés pour exclure automatiquement les régions de la séquence *requête* qui ont une faible complexité en composition, ou qui ont des répétitions de faible période. La présence de telles séquences peut mener à un grand nombre de MSP statistiquement significatifs, mais biologiquement inintéressants.

BLAST contient 4 programmes :

- BLASTP : recherche entre une séquence protéique et une banque de séquences protéiques.
- BLASTN : recherche entre une séquence nucléique et une banque de séquences nucléiques.
- TBLASTN : recherche entre une séquence protéique et une banque de séquences nucléiques, en regardant les 6 phases[7] pour chaque séquence de la base de données.
- BLASTX : recherche entre une séquence nucléique et une banque de séquences protéiques, en regardant les 6 phases pour la séquence *requête*.

BLAST est fondé sur des propriétés statistiques très importantes. Soit deux séquences aléatoires composées de lettres iid[8]. Supposons que l'espérance du score entre deux lettres quelconques soit négative et que la probabilité d'avoir un score strictement positif entre deux lettres ne soit pas nulle, alors le meilleur score suit une loi de Gumbel, loi des valeurs extrêmes avec les paramètres $K$ et $\lambda$ [78]. Ces paramètres interviennent dans le calcul de la signification statistique du meilleur score.

Considérons tout d'abord une suite $(X_i)_{i\in\mathbb{N}}$ de variables aléatoires iid, de distribution $X$. Soit $S_k$ les sommes partielles : $S_k = \sum_{i=1}^{k} X_i$. Le théorème suivant donne la distribution de $M(n) = max_{1\leq j\leq k\leq n}(S_k - S_j)$.

**Théorème 1.3.1** *(Karlin, Dembo et Kawabata, 1990, [79]) Soit une variable aléatoire $X$ telle que $E(X) < 0$ et telle que $P(X > 0) > 0$. Soit $X_n$ une suite de variables aléatoires indépendantes et distribuées comme $X$. Soient les sommes partielles $S_k = \sum_{i=1}^{k} X_i$ et $M(n)$ la section maximale de la marche aléatoire $(S_n)$, définie par $M(n) = \max_{1\leq i\leq j\leq n}\{S_j - S_i\}$. La queue de distribution de $M(n)$ est donnée par la formule*

$$P\Big(M(n) > \frac{log(n)}{\lambda} + x\Big) = 1 - e^{-Ke^{-\lambda x}}$$

*où la constante $\lambda$ est la solution positive de l'équation en $\xi$ :*

$$E[e^{\xi X}] = 1$$

*et la constante $K$ est donnée par la formule*

$$K = \frac{exp\Big\{-2\sum_{k=1}^{\infty}\frac{1}{k}\Big(E(e^{\lambda X}; S_k < 0) + Pr(S_k \geq 0)\Big)\Big\}}{\lambda E(Xe^{\lambda X})}.$$

6. National Center for Biotechnology Information, unité du National Institutes of Health (NIH)

7. Chaque phase de lecture correspond à un choix du début du codon initial dans le brin d'ADN. Il y a trois phases de lecture pour chaque sens de lecture (Cf. Annexe A).

8. Indépendantes Identiquement Distribuées

Cette formule peut être appliquée à la comparaison de séquences 2 à 2, en assignant des scores aux paires de lettres et en remplaçant N par $n \times m$, produit des longueurs des deux séquences à comparer.

De plus si on pose $y = K \times exp(-\lambda \times S)$, la probabilité de trouver $k$ alignements distincts, chacun de score supérieur à $S$, est donnée par :

$$P(M_1, M_2 \ldots M_k > S) = 1 - \left[\sum_{i=0}^{k-1} \frac{y^i}{i!}\right] exp(-y) \ .$$

### 1.3.2 $2^{ème}$ heuristique : FASTA

FASTA est un programme de comparaison de séquences orienté vers la comparaison d'une séquence *requête* contre une banque de séquences.

Le programme FASTA [89, 105] est fondé sur une heuristique permettant de diminuer le temps de la recherche dans la base. Mais, comme dans beaucoup de cas, l'augmentation de la sensibilité se fait au détriment de la rapidité.

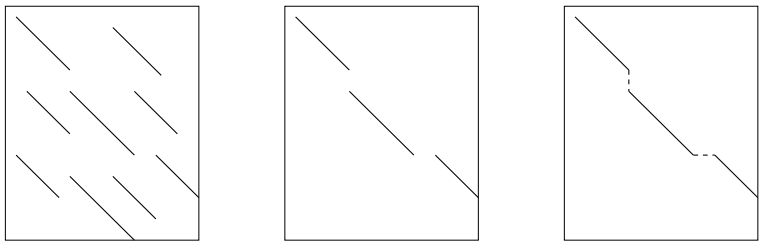



FIG. 1.8 - **Les étapes de l'algorithme FASTA.** *La première étape consiste en la recherche des zones de fortes similarités. On cherche ensuite à relier ces zones entre elles, en utilisant des pénalités pour les gaps. Enfin, on utilise la programmation dynamique sur des sous-séquences.*

L'algorithme de FASTA se décompose en 5 étapes :

1. L'algorithme établit tout d'abord la liste exhaustive de tous les mots de longueur *ktup* qui existent dans les deux séquences. Le paramètre *ktup* détermine la sensibilité de l'algorithme. Une table de hachage est mise en oeuvre permettant le stockage pour la séquence *requête* de toutes les positions où chaque k-uplet apparaît. Ainsi, lorsqu'on parcourt une séquence de la banque, on a immédiatement la position de toutes les occurrences dans la séquence requête du k-uplet courant.

   En général pour les protéines le paramètre *ktup* est positionné à 2. Pour l'ADN, sa valeur est de 6. L'utilisateur peut modifier ce paramètre pour augmenter la sensibilité. Ceci se fera toutefois au détriment du temps de calcul.

2. Détermination des zones denses en identité : l'algorithme détermine ensuite les diagonales qui contiennent le plus de k-uplets en commun.

3. Calcul de *init*1 : en fonction de la matrice de substitution, l'algorithme calcule pour toutes les diagonales retenues, le score d'alignement. On ne retient que les 10 meilleures régions.

4. Au cours de la quatrième étape, FASTA teste si les régions aux scores les plus élevés peuvent être reliées entre elles. Ainsi, s'il y a plusieurs scores plus grands que *Cutoff*, on cherche à joindre les différentes régions, en permettant des insertions/délétions, et ceci suivant un schéma déterminé de pénalités. Le score *initn* est calculé en soustrayant les coûts des insertions/délétions à la somme des scores des différentes régions. Le score *initn* est utilisé pour classer par ordre décroissant les différents alignements dans la banque.

5. Enfin, la programmation dynamique est utilisée uniquement sur un voisinage de l'alignement correspondant à *initn*, sur une bande de largeur 64. L'algorithme de Smith-Waterman donne alors le score final *opt*.

FASTA cherche à obtenir l'alignement le plus long possible et ce en jouant sur les insertions/délétions. Il est à remarquer que la recherche par mot (*ktup*) fait que l'algorithme est moins sensible en ce qui concerne les séquences légèrement divergentes. De plus, le fait de chercher un seul score maximum peut entraîner l'élimination de régions de forte similarité, mais dont le score est insuffisant pour apparaître au tableau final. Si deux régions de très fortes homologies existent, seule celle de plus haut score apparaîtra.

```
 28     0     0 :*
 30     0     1 :*
 32     5     6 :===*
 34    16    16 :========*
 36    34    32 :================*=
 38    65    53 :============================*======
 40    82    74 :=======================================*=====
 42    68    90 :==================================          *
 44   102   100 :==================================================*=
 46    97   101 :================================================== *
 48    96    97 :==================================================*
 50    85    89 :========================================== *
 52    73    78 :=================================== *
 54    84    66 :===================================*=========
 56    51    56 :========================= *
 58    36    46 :==================    *
 60    29    37 :================   *
 62    25    30 :============= *
 64    22    24 :===========*
 66    20    19 :==========*
 68    14    15 :========*
 70    19    11 :======*=====
 72     9     9 :=====*
 74    12     7 :===*==
 76     9     5 :==*==
 78     8     4 :=*==
 80     7     3 :=*==
 82     3     3 :=*
 84     2     2 :*
 86     5     2 :*==
 88     0     1 :*
 90     0     1 :*
 92     1     1 :*
 94     0     1 :*
 96     0     0 :*
 98     0     0 :*
100     0     0 :*
334060 residues in  1082 sequences
statistics extrapolated from 1079 to 1079 sequences
results sorted and z-values calculated from opt score
1082 scores better than 1 saved, ktup: 2, variable pamfact
BLOSUM50 matrix, gap penalties: -12,-2
joining threshold: 36, optimization threshold: 24, width: 16
```

FIG. 1.9 - **Histogramme construit par FASTA :** *On compare la séquence AHRC_BACSU avec toutes les séquences de *_BACSU de Swissprot. Les «=» représentent l'histogramme obtenu à partir des scores* initn. *Les «*» modèlisent la loi de Gumbel estimée sur les données.*

Après avoir analysé toutes les séquences de la banque de données, FASTA imprime les scores *initn* de chaque séquence dans un histogramme (Cf. figure 1.9). A partir de la moyenne des scores *initn* et de l'écart-type empiriques, FASTA calcule les paramètres de la loi de Gumbel qui modèlise le mieux les scores obtenus. Dans la figure 1.9, l'histogramme des données est représenté par des «=», et les effectifs des classes sont donnés dans la deuxième colonne. L'histogramme de la loi de Gumbel associée est représenté par les «*» et les effectifs de chaque classe sont donnés par la troisième colonne. De plus FASTA donne, à partir des scores *opt*, le score centré-réduit appelé le Z-score.

Si $initn > init1$, cela signifie qu'il y a au moins deux régions dans la séquence de la banque similaires à la séquence recherchée et que ces régions sont assez proches les unes des autres pour avoir été raccrochées par l'algorithme.

Si $opt > initn$, cela signifie que les régions homologues sont améliorées par l'addition de insertions/délétions dans une des séquences ou dans les deux.

## 1.4 Alignements multiples

L'alignement de $n$ séquences (ou alignement multiple) est un problème NP-difficile (quel que soit $n$). Le problème de la recherche d'une sous-séquence commune de longueur maximale [47] est déjà un problème NP-difficile.

En fait, la définition de la fonction de score d'un alignement n'est pas aussi simple que dans le cas de deux séquences. La fonction la plus communément utilisée repose sur le modèle SP-score (*Sum of Pair*) qui définit le score d'alignement comme la somme des scores des alignements par paire.

### 1.4.1 Programmation dynamique: généralisation de l'algorithme SW

Pour comparer plusieurs séquences, Needleman & Wunsch [101] ont suggéré une extension de la programmation dynamique qui a été utilisée par Murata et al. [99] pour la comparaison de trois séquences. Waterman et al. [130] ont décrit la façon dont la programmation dynamique peut être appliquée dans le cas plus général de l'alignement de $n$ séquences.

La généralisation de la programmation dynamique nécessite quelques définitions. Soit $\mathcal{S} = (S_1, S_2, \ldots, S_n)$, l'ensemble des séquences à aligner.

- Notons $\vec{i} = (i_1, i_2, \ldots, i_n)$, un vecteur de $\mathbb{N}^n$.
- Notons $C(\vec{i}) = C(i_1, i_2, \ldots, i_n)$ le coût minimal de l'alignement des préfixes $S_k[1, i_k]$ de longueur $i_k$ de chaque séquence $S_k$, pour $k = 1, 2, \ldots, n$.
- Pour l'alignement de deux séquences, on procède par colonne de l'alignement: on rajoute une paire d'alignement (du type $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$, $\begin{bmatrix} a_i \\ - \end{bmatrix}$ ou $\begin{bmatrix} - \\ b_j \end{bmatrix}$) à chaque itération. Ici la hauteur de la colonne ne sera pas 2 mais $n$. La coût d'édition correspondant à une colonne ayant les lettres ou «$-$» $(a_1, a_2, \ldots, a_n)$ sera donné en vertu du modèle SP-score par:

$$d(a_1, a_2, \ldots, a_n) = \sum_{1 \leq i < j \leq n} d(a_i, a_j) \tag{1.17}$$

où $d(a_i, a_j)$ est le coût d'édition habituel pour l'alignement de deux séquences.

La récurrence centrale de la programmation dynamique s'écrit :

$$C(\vec{i}) = max\Big\{C(\vec{i}-\vec{e}) + d\Big(f(e_1, S_1[i_1]), f(e_2, S_2[i_2]), \ldots, f(e_n, S_n[i_n])\Big)\Big\} \tag{1.18}$$

$$avec\ \ f(e_k, S_k[i_k]) = \begin{cases} S_k[i_k] & si\ \ e_k = 1 \\ \text{«}-\text{»} & sinon\ . \end{cases}$$

Le maximum est pris sur l'ensemble des vecteurs $\vec{e}$ de $\{0,1\}^n$ qui ont au moins une coordonnée non nulle.

Il est remarquable que ce problème peut se voir encore comme un problème de graphe :

- L'ensemble des sommets est l'ensemble des vecteurs de $\mathbb{N}^n$ dont la valeur de la i$^{\text{ème}}$ coordonnée ne dépasse pas la longueur de la i$^{\text{ème}}$ séquence.
- Il existe un arc du sommet $\vec{I_1}$ vers le sommet $\vec{I_2}$ si et seulement si $\forall i \in \{1, 2, \ldots, n\}$, $0 \leq I_2[i] - I_1[i] \leq 1$. Chaque sommet (dans l'intérieur du graphe) est atteint par $(2^n - 1)$ arcs. De chaque sommet partent $(2^n - 1)$ arcs.
- La valeur associée à un arc entre les sommets $\vec{I_1}$ et $\vec{I_2}$ est :

$$d\begin{pmatrix} f(\vec{I_2}[1] - \vec{I_1}[1], S_1[i_1]), \\ f(\vec{I_2}[2] - \vec{I_1}[2], S_2[i_2]), \\ \vdots \\ f(\vec{I_2}[n] - \vec{I_1}[n], S_n[i_n]) \end{pmatrix} \tag{1.19}$$

$$avec\ \ f(I[k], S_k[i_k]) = \begin{cases} S_k[i_k] & si\ \ I[k] = 1 \\ \text{«}-\text{»} & sinon\ . \end{cases}$$

- Le problème est de trouver les deux sommets tels que le chemin allant de l'un à l'autre ait un coût maximum.

Malheureusement, le temps de calcul de la matrice $C$ (Cf. éq. 1.18) croît exponentiellement par rapport au nombre de séquences. Le nombre de sommets est $\prod_{i=1}^{n} l_i$. L'algorithme est en $O\Big((2^n - 1).\prod_{i=1}^{n} l_i\Big)$.

### 1.4.2 Alignement multiple par groupement d'alignements par paire

La méthode générale est la construction itérative de l'alignement multiple, en groupant 2 à 2 les alignements par paire, sélectionnés parmi toutes les paires de séquences possibles. Trois étapes sont à distinguer :

- la méthode d'alignement par paire de séquences,
- l'ordre des regroupements des alignements,
- l'alignement lui même entre les alignements par paire.

Cette approche est la plus répandue : les outils mettant en œuvre ces trois étapes sont très nombreux. D'autre part les différentes étapes précédentes ne sont pas toujours bien expliquées, il en résulte parfois quelques confusions. Notre but n'étant pas d'être exhaustif, nous ne citerons que trois programmes d'alignements multiples fonctionnant par regroupement d'alignements : les programmes Clustal, ClustalW et TreeAlign.

### 1.4.3 Clustal

CLUSTAL[64] a été certainement l'un des outils d'alignements multiples par regroupement d'alignements les plus utilisés. Décrivons rapidement les trois étapes précédentes.

- La méthode d'alignement par paire de séquences est celle de Wilbur et Lipman [134], une méthode rapide mais approximative. Le score est en fait le nombre d'alignements exacts de k-uplets ($k = 1$ ou 2 pour les protéines, $k = 2, 3$ ou 4 pour les séquences d'ADN) dans le meilleur alignement auquel on a retranché une pénalité pour chaque insertion/délétion.

- L'ordre des regroupements des alignements est donné par la méthode UPGMA [118]. L'ordre d'agrégation est guidé par un arbre explicitement calculé avant le regroupement des alignements. Feng et Doolittle [41] proposent de regrouper les deux séquences les plus proches, puis de recalculer ensuite une matrice de dissimilarité qui contient les distances moyennes entre les deux séquences choisies et chacune des séquences restantes. Le processus est ainsi répété tant que toutes les séquences ne sont pas agrégées.

- L'alignement lui-même entre les alignements repose sur des séquences consensus. On considère qu'un alignement est un mot dont les caractères sont les colonnes de l'alignement [124] [33] [64]. On crée à chaque itération une séquence *consensus*, qui est construite en retenant, à chaque position, uniquement la lettre la plus fréquente. L'alignement de deux alignements devient simplement un alignement de deux séquences.

### 1.4.4 ClustalW

Le passage de Clustal à ClustalW [125] a été l'occasion d'un changement de stratégie sur les trois points qui définissent la méthode.

- Le programme laisse à l'utilisateur le choix de la méthode d'alignement par paire: soit la méthode de Wilbur et Lipman[134] utilisée dans Clustal, soit la programmation dynamique de Smith & Waterman.

- L'arbre utilisé pour agréger les séquences au fur et à mesure est construit à partir de la matrice des «*distances*» calculées dans la phase précédente, par la méthode des plus proches voisins (*Neighbor-Joining* [108]).

- La programmation dynamique est utilisée, mais, on n'a plus à faire à une séquence consensus qui modélise un alignement. Ici, chaque coût de substitution dans la programmation dynamique est la moyenne de tous les coûts de substitution. S'il y a une insertion/délétion à la position considérée dans l'un des alignements construits au préalable, on n'en tient pas compte pour le calcul du coût de substitution.

### 1.4.5 TreeAlign

La méthode développée par Hein [60, 62, 63] consiste à chercher la phylogénie et à aligner les séquences selon la phylogénie. L'approximation centrale consiste à décomposer le problème de l'alignement multiple en sous-problèmes d'alignements deux à deux et à construire à chaque étape des séquences consensus supposées être les séquences ancêtres. A chaque étape l'ensemble des séquences ancêtres les plus parcimonieuses est traité. On utilise une généralisation de l'algorithme de SW qui permet la comparaison de graphes dont les arcs correspondent à des ensembles d'acides aminés.

Lors de la phase de remontée de l'algorithme de programmation dynamique, un ensemble de séquences est généré. Cet ensemble est supposé être composé des ancêtres possibles communs aux deux séquences en présence. La programmation dynamique est généralisée pour pouvoir comparer des graphes de séquences. Un graphe de séquences est équivalent à l'ensemble des alignements optimaux entre plusieurs séquences. La recherche de l'alignement de deux graphes d'alignement se fait par programmation dynamique, et mène directement à un alignement multiple. Voici les trois points caractérisant TreeAlign.

- On calcule les scores d'alignements par paire grâce à la programmation dynamique.
- Pour la construction d'un premier arbre, la méthode utilisée par Hein est décrite dans [61]. Après cette phase, l'algorithme réarrange l'arbre. La méthode la plus simple est la permutation des plus proches voisins tant que le poids total de l'arbre diminue.
- L'arbre ainsi construit est utilisé comme guide pour l'alignement multiple. C'est à cet endroit que la modification de la programmation dynamique est utilisée. A ce stade, on refait des arrangements pour rendre encore plus parcimonieux l'arbre déjà construit.

## 1.5 Variations sur l'algorithme de Smith & Waterman

### 1.5.1 Diminution de l'espace mémoire : récursion de Hirschberg

La méthode de la programmation dynamique calcule le score d'alignement optimal en un temps proportionnel au produit des longueurs des deux séquences. Mais, dans la version la plus simple, l'alignement n'est pas explicitement calculé, puisque seuls le score et la position de la fin de l'alignement sont donnés.

Rappelons que la méthode la plus simple pour calculer l'alignement est de parcourir la matrice entière en sens contraire. Une fois qu'on a trouvé la position de la fin de l'alignement, on peut, pour chaque case, retrouver la case qui a permis d'obtenir le meilleur score (Cf. figure 1.4, p. 27). La complexité en espace de cet algorithme est $O(m \times n)$ où $m$ et $n$ sont les longueurs respectives des deux séquences. Par conséquent, le traitement des séquences longues pose un problème de place mémoire. L'introduction de méthodes du type de celle présentée par Hirschberg [66] semble nécessaire. Miller et Myers [93] ont appliqué cette méthode pour l'alignement de séquences. Le principe est simple, c'est celui de *diviser pour régner*.

Il s'agit d'un algorithme récursif qui divise la première séquence en deux, puis cherche le résidu de la deuxième séquence qui permet d'aligner au mieux le début de la première séquence avec la deuxième séquence tronquée au résidu cherché. Ainsi, les paires de résidus alignés sont trouvées récursivement, de part et d'autre du point central, jusqu'à ce que tous les résidus de la première séquence soient alignés.

Soit deux séquences $A$ et $B$ de longueur respective $n$ et $m$. L'algorithme se décompose en six phases (Cf. figure 1.10) :

1. On commence par diviser en deux la séquence A : on s'intéresse au résidu $i^* = \lfloor \frac{n}{2} \rfloor$ (Cf. figure 1.10, étape 1).
2. Pour tous les indices $j$, on stocke les scores d'alignements de $A[1, i^*]$ avec $B[1, j]$ (Cf. figure 1.10, étape 2). Pour chaque case, deux scores sont mémorisés :
   - les scores des chemins finissant par un appariement ou par une insertion/délétion,

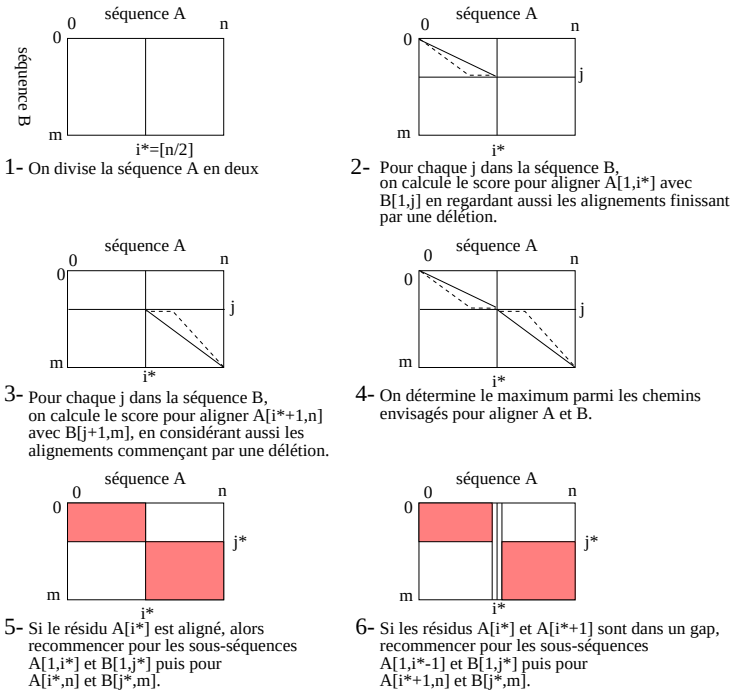



FIG. 1.10 - **Principe de Hirschberg :** *récursivement, on détermine les appariements successifs.*

- les scores des chemins finissant par une délétion.

3. De la même manière, pour tous les indices $j$, on stocke les scores d'alignements de $A[i^*+1, n]$ avec $B[j+1, m]$ (Cf. figure 1.10, étape 3). On mémorise :
   - les scores des chemins commençant par un appariement ou par une insertion/délétion,
   - les scores des chemins commençant par une délétion.

4. Le chemin optimal pour aligner les séquences entières passe forcément par la colonne $i^*$. On peut donc le déterminer en recherchant le maximum parmi toutes les sommes entre un score de l'étape 2 et un score de l'étape 3 (Cf. figure 1.10, étape 4). Lorsqu'il y a une délétion, il faut faire attention à ne pas compter deux fois une pénalisation d'ouverture de gap.

   Soit $j^*$ le résidu de B tel que le chemin optimal passe par $(i^*, j^*)$.

   Les étapes 2, 3 et 4 ont une complexité mémoire en $O(m)$, Cette complexité est linéaire.

5. Si le chemin optimal passe par un appariement entre les lettres $A[i^*]$ et $B[j^*]$, on recommence les étapes 1 à 4, pour les sous-séquences $A[1, i^*]$ et $B[1, j^*]$ puis pour $A[i^*, n]$ et $B[j^*, m]$ (Cf. figure 1.10, étape 5). De plus, on mémorise que la lettre $A[i^*]$ est alignée avec $B[j^*]$.

6. Si les résidus $A[i^*]$ et $A[i^*+1]$ sont dans un gap, on recommence les étapes 1 à 4, pour les sous-séquences $A[1, i^*-1]$ et $B[1, j^*]$ puis pour $A[i^*+1, n]$ et $B[j^*, m]$ (Cf. figure 1.10, étape 6).

Pas à pas, on reconstruit l'alignement, puisqu'à chaque étape, on stocke un appariement, ou une délétion. L'espace nécessaire pour les étapes 5 et 6, peut être récupérer dans l'espace utilisé pour les étapes 1 à 4. Globalement, la complexité en espace est $O(m + log_2(n))$. En effet, il faut l'espace pour mémoriser les scores intermédiaires (deux colonnes de taille $m$) et il faut mémoriser à chaque étape soit un appariement, soit une délétion. Le nombre de découpage est de l'ordre de $k = log_2(n)$.

Cet algorithme permet de réduire l'espace mémoire nécessaire pour le calcul de l'alignement, au prix d'une augmentation du temps d'exécution. En effet, lorsqu'on passe à l'étape 5, la zone hachurée est exactement la moitié de la zone initiale. Dans le cas de l'étape 6, la zone hachurée est légèrement inférieure. Le temps nécessaire pour déterminer les deux paires d'alignements suivantes est donc la moitié du temps nécessaire pour déterminer la première paire. Il s'ensuit que la complexité totale est en

$$O\Big(m \times n \times \underbrace{(1 + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} ...)}_{A}\Big).$$

Puisque le nombre de découpage est de l'ordre de $k = log_2(n)$, le terme $A$ vaut $\frac{2^k - 1}{2^{k-1}} < 2$ et la complexité temporelle est de l'ordre de $O(2 \times m \times n)$.

Il est à remarquer que cette méthode nécessite la connaissance des bornes de l'alignement local, ce qui est calculable par l'exécution de l'algorithme de SW sur les séquences lues, en sens inverse, à partir des coordonnées de la fin de l'alignement.

### 1.5.2 Les K-meilleurs alignements

L'algorithme de Smith & Waterman peut être modifié pour permettre l'identification de tous les alignements locaux optimaux et sous-optimaux, qui n'ont aucun sous-alignement en commun. Les premiers à avoir développé un tel algorithme [132] ont utilisé un algorithme itératif nécéssitant un espace important ($O(m \times n)$, où $m$ et $n$ sont les longueurs des deux séquences).

#### Algorithme de declumping

Pour choisir l'ordre d'apparition des alignements, les auteurs définissent un ordre total sur les scores de la matrice de la programmation dynamique. Lors de la comparaison de deux séquences, il y a en général plusieurs alignements optimaux. Waterman et Eggert [132] énumèrent les alignements optimaux dans l'ordre suivant :

- Si $S(i,j) = S(k,l)$ et $i + j < k + l$, alors l'alignement finissant en $(i,j)$ sera énuméré en premier.
- Si $S(i,j) = S(k,l)$ et $i + j = k + l$ et $i < k$, alors l'alignement finissant en $(i,j)$ sera énuméré en premier.

Deux alignements seront dit **chevauchants** s'ils ont un appariement en commun, ou s'ils partagent une insertion/délétion. La manière la plus simple pour calculer les alignements sous-optimaux est de recalculer les valeurs de la matrice, en ne permettant aucun appariement intervenant dans l'alignement précédemment calculé. Supposons qu'on ait déterminé un alignement optimal commençant en $(i^*, j^*)$. Pour le calcul de l'alignement sous-optimal, non-chevauchant avec le précédent, seules les valeurs de la matrice pour $i \geq i^*, j \geq j^*$ doivent être recalculées, c'est-à-dire les valeurs de la zone non hachurée dans la figure 1.11. En fait, il y a beaucoup moins de valeurs à recalculer. Seules les valeurs *proches* de l'alignement précédemment exhibé doivent être modifiées. Considérons

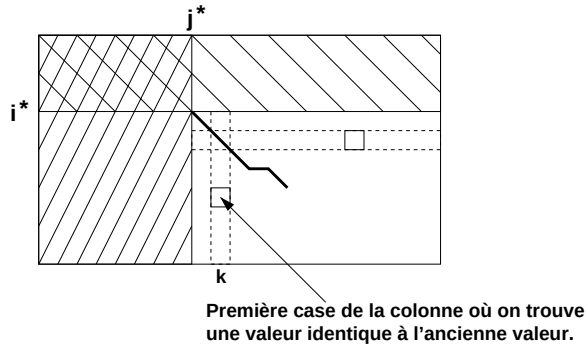



FIG. 1.11 - **Algorithme de declumping : zone à recalculer**

une colonne $(k, j)$ pour $j \geq j^*$, dont tous les éléments devraient être recalculés (Cf. figure 1.11). On calcule les nouvelles valeurs pour $(k, j)$ pour $k = i^*, k = i^* + 1, k = i^* + 2, \ldots$ jusqu'à ce que les nouvelles valeurs soient égales aux anciennes. A partir de là, il est clair que les valeurs de la colonne ne changeront plus. Il en est de même pour les lignes.

Ainsi, les k-meilleurs alignements sont calculés itérativement, sans pour autant avoir à spécifier le nombre voulu d'alignements. Il est à remarquer qu'on doit stocker l'ensemble de la matrice de la programmation dynamique, ce qui rend cet algorithme coûteux en espace ($O(m \times n)$). La complexité en temps est cependant bien réduite.

Pour Huang et Miller [95], cet algorithme a la propriété de non-unicité, c'est-à-dire qu'on peut trouver deux listes de k-alignements optimaux et sous-optimaux, telles que les scores des deux $l^{ièmes}$ alignements, diffèrent. L'argumentation fait intervenir le choix du début de l'alignement lorsqu'il existe plusieurs débuts possibles donnant le même score. Ce qui est important, ce n'est pas tellement le premier sommet de l'alignement, mais les règles de décision lors de la remontée. On pourra se référer à la section 2.7.

**Algorithme linéaire en espace :**

Huang et al. [94] ont développé un algorithme linéaire en espace mais relativement lent puisque sa complexité temporelle est proportionnelle à $k.m.n$, pour avoir les k-meilleurs alignements. Huang et Miller [95] ont amélioré cet algorithme et l'espace nécessaire devient en $O(m+n+K)$, où $m$ et $n$ sont les longueurs des séquences et $K$ est la longueur cumulée des k-meilleurs alignements calculés.

L'approche des auteurs est la suivante : l'algorithme met en œuvre une boucle de programmation dynamique comme celle de l'algorithme de Smith & Waterman, mais en stockant les k-meilleurs alignements, au lieu de ne garder que le meilleur. Ils retiennent les valeurs $S(i_1, j_1)$ et $S(i_2, j_2)$ uniquement si les alignements associés correspondent à des alignements qui ne se chevauchent pas, c'est-à-dire à des chemins disjoints dans le graphe associé. L'algorithme exécute une passe complète sur l'ensemble du graphe pour trouver *le* meilleur alignement, tout en gardant les k meilleurs scores. Une fois le chemin optimal calculé, le problème est de découvrir les autres chemins de haut score, qui avaient été cachés par la première passe. Pour chaque alignement supplémentaire, l'algorithme exécute une passe de la programmation dynamique *backward* et une passe *forward*. La phase *backward* s'exécute sur une zone limitée et permet de déterminer la «*zone d'influence*» de l'alignement précédemment calculé. La phase *forward* recalcule les scores SW sur cette région. Au cours de la deuxième passe, l'algorithme met à jour la liste des k-meilleurs alignements locaux.

**Stockage du début de l'alignement :**

La programmation dynamique peut être modifiée pour avoir le début de l'alignement sans augmenter l'ordre de grandeur de la complexité. Puisqu'il peut se faire qu'il y ait plusieurs alignements possibles ayant le même score et terminant au même endroit (Cf. section 1.2.2, figure 1.5), il faut choisir un début de l'alignement. Pour cela, on introduit un ordre parmi les doubles indices $(i, j)$, qui peut être aussi bien celui du parcours des indices qu'un autre ordre. Chaque case hérite du début de l'alignement de la case qui a permis d'obtenir le maximum. Pour être plus précis, dans le cas d'une pénalité linéaire des insertions/délétions, l'instruction du type :

```
M[i][j] = max ( M[i-1][j-1] + s[A[i]][B[j]] ,
                M[i-1][ j ] - gap           ,
                M[ i ][j-1] - gap           ,
                0 );
```

correspondant à l'équation 1.13, doit être complétée par les instructions suivantes :

```
if (M[i][j] == M[i-1][j-1] + s[A[i]][B[j]] ) then
       DEBUT[i][j] = DEBUT[i-1][j-i];
else if (M[i][j] == M[i-1][ j ] - gap      ) then
       DEBUT[i][j] = DEBUT[i-1][ j ];
else if (M[i][j] == M[ i ][j-1] - gap      ) then
       DEBUT[i][j] = DEBUT[ i ][j-i];
else if (M[i][j] == 0                      ) then
       DEBUT[i][j] = (i,j);
```

où *DEBUT* est une structure représentant les indices de début d'alignement. La généralisation au cas des pénalités affines est immédiate.

En utilisant cette remarque, Barton a donné un algorithme déterminant les meilleurs alignements locaux en une seule passe [18]. Cet algorithme repose sur la remarque que l'on ne cherche que des alignements sous-optimaux qui ne se chevauchent pas. Ainsi, lors de la phase de descente, on peut maintenir à la fois le score obtenu en chaque case, mais aussi le début de l'alignement ayant permis ce score, par le même type d'instructions. Cet algorithme est toujours quadratique en temps et linéaire en espace.

Cependant, cet algorithme peut passer à côté de certains alignements significatifs. La figure 1.12 illustre un des cas où l'algorithme de Barton oublie de rapporter des alignements sous-optimaux.

### 1.5.3 Fonctions de pénalisation des insertions/délétion concaves

Supposons que deux protéines ont une structure tertiaire semblable mise à part une longue boucle dans l'une des deux séquences qui n'a pas son équivalent dans l'autre. L'alignement sur les séquences primaires fait intervenir une très longue insertion/délétion. En dehors de cette boucle, les deux séquences peuvent être très proches. Si la boucle est très longue, l'algorithme de Smith & Waterman risque de ne pas retrouver cet alignement, puisque le coût d'une très longue insertion/délétion est trop fort. Trois cas peuvent se présenter :

- L'algorithme préfère aligner deux sous-séquences, qui ne correspondent à aucune partie du «*vrai alignement*».

- L'algorithme trouve uniquement une des parties de l'alignement souhaité.

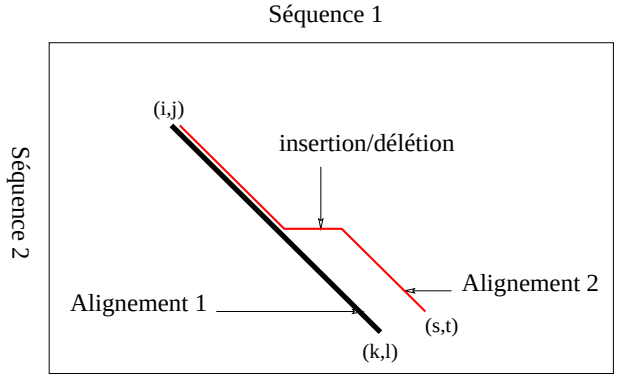



FIG. 1.12 - **Alignements sous-optimaux :** *l'algorithme de Barton ne repère pas toutes les zones de similarité. Une zone de haute similarité peut cacher une zone de plus faible similarité. Lors de la phase de descente, l'algorithme considère que les alignements* 1 *et* 2 *(se terminant en* $(k,l)$ *et* $(s,t)$ *respectivement) partagent le début de leurs alignements et commencent tous deux en* $(i,j)$*. L'algorithme ne retient que l'un des deux, celui de plus grand score. L'alignement* 2 *est ignoré puisqu'il appartient au domaine d'attraction de l'alignement* 1*.*

– Il trouve le bon alignement, mais comme il a créé une très longue insertion/délétion, le score obtenu, n'est pas significatif de la similarité réelle.

Dans de tels cas, la pondération des insertions/délétions n'est pas adaptée. On aurait envie de pénaliser les insertions/délétions de manière à permettre l'apparition de très longs gaps. Biologiquement, l'opération qui coûte cher est la création d'une insertion/délétion. Plus elle est longue, plus le surcoût dû à son allongement est faible.

La fonction de coût pour une insertion/délétion de longueur $k$ doit être concave. Dans de tels cas, Miller et Myers [92] ont repris l'algorithme de Waterman [131], et ont proposé deux algorithmes de programmation dynamique plus performants qui répondent efficacement au problème. Les deux approches maintiennent $N+2$ listes de *candidats*, où $N$ est la longueur de l'une des deux séquences. La liste de *candidats* représente, par exemple pour une ligne, à la position $k$ dans la ligne, l'ensemble des positions antérieures à $k$, qui sont susceptibles d'intervenir pour les insertions/délétions suivantes.

Ces méthodes sont de complexité en temps $O(n^2.log(n))$ dans le pire des cas, et $O(n^2)$ sous certaines conditions. C'est le cas lorsqu'il existe une solution analytique de l'équation en $k$, pour $x > 0$ :

$$g(k) = g(k-x) + y$$

### 1.5.4 Méthode dépendant du contexte : l'algorithme de Huang

Sellers [111] a introduit une métrique où les substitutions, les délétions et les insertions sont indépendantes du contexte. Waterman et al. [130] ont généralisé la métrique de Sellers en rendant les délétions et insertions dépendantes du contexte. Wilbur et Lipman [135] ont présenté un cadre général pour l'évaluation des substitutions à contexte dépendant. La raison principale pour laquelle on cherche à introduire une dépendance en fonction du contexte, est que les mutations

biologiques apparaissent de façon non-uniforme le long des séquences [87]. Cela entraîne une distribution non uniforme des *appariements exacts* dans les alignements de séquences : il y a des régions ou les appariements exacts sont plus concentrés. Au contraire, si on prend des séquences aléatoires uniformément distribués et indépendantes, on se rend compte que les appariements exacts sont uniformement distribués [71]. Cette observation doit être introduite dans la recherche de la signification d'un alignement : un alignement comportant 5 appariements exacts consécutifs est beaucoup plus pertinent qu'un alignement comportant 5 appariements exacts dispersés. Un «*bonus*» doit être donné à de tels alignements.

Huang présente un algorithme prenant en compte le contexte en valorisant tout appariement exact si celui-ci se trouve dans une région de l'alignement où les acides aminés sont bien conservés. Il développe un algorithme aussi fondé sur la programmation dynamique, qui répond à la question avec une complexité $O(m \times n)$ en temps et en $O(m + n)$ en espace. Pour calculer l'alignement, il utilise aussi le schéma de Hirschberg, comme l'avaient fait Myers et Miller [93].

**Le schéma de score :**

Rappelons que l'on ne considère que trois types de paires alignées :

1) Les appariements : $\begin{bmatrix} a \\ b \end{bmatrix}$. Ils sont à nouveau séparés en deux classes : les appariements exacts appelés aussi identités ou «matches» ($a = b$) et les appariements non-exacts ou substitutions ( en anglais «mismatches» ) ($a \neq b$). Un *bloc de substitutions* est une sous-suite d'appariements délimitée par des paires de délétions ou d'insertions ou par la fin de la séquences.

2) Les paires d'insertions $\begin{bmatrix} - \\ b \end{bmatrix}$. Une suite de paires d'insertions est appelée une insertion de longueur égale au nombre de paires d'insertion.

3) Les paires de délétions $\begin{bmatrix} a \\ - \end{bmatrix}$. Une suite de paires de délétions est appelée une délétion de longueur égale au nombre de paires de délétion.

Huang se place dans le cas des coûts affines d'insertions/délétions. Son algorithme consiste à donner un bonus aux *appariements exacts* lorsque dans un proche voisinage, il y a d'autres *matches*. Un *match* reçoit un *bonus à droite* d'une valeur de $\delta(a, a)$ s'il existe une autre *match* à moins de $l$ positions à droite de la position considérée dans le même bloc de substitutions. De la même manière, un *match* reçoit un *bonus à gauche* d'une valeur de $\delta(a, a)$ s'il existe un autre *match* à moins de $l$ positions à gauche de la position considérée dans le même bloc de substitutions. La constante $l$ est appelée *la longueur de la dépendance*.

Le score de l'alignement est la somme des scores indépendamment du contexte, plus la somme des bonus pour chaque *match* de l'alignement.

**Alignement global :**

Notons $c(A[i-k+1, i], B[j-k+1, j])$ le score du bloc de substitutions entre les sous-séquences $A[i-k+1, i]$ et $B[j-k+1, j]$, où $k \leq min(i, j)$. $c(A[i-k+1, i], B[j-k+1, j])$ est égal à la somme des coûts de substitution plus la somme des bonus pour chacun des *matches* du bloc de substitution. Notons $S(i, j)$ le score d'un alignement global optimal de $A[1, i]$ et $B[1, j]$. Soit $H(i, j)$ le score maximum pour un alignement des sous-séquences $A[1, i]$ et $B[1, j]$ terminant par

l'appariement $\begin{bmatrix} a_i \\ b_j \end{bmatrix}$. Le calcul du score se fait par la récurrence suivante:

$$S(i,j) = \begin{cases} max(H(i,j), E(i,j), F(i,j)) & si\ \ i>0\ \ et\ \ j>0 \\ -(go + ge \times (i-1)) & si\ \ i>0\ \ et\ \ j=0 \\ -(go + ge \times (j-1)) & si\ \ i=0\ \ et\ \ j>0 \\ 0 & si\ \ i=0\ \ et\ \ j=0 \end{cases} \quad (1.20)$$

$$H(i,j) = \max_{1 \le k \le min(i,j)} \begin{pmatrix} S(i-k, j-k) \\ + \\ c(A[i-k+1, i], B[j-k+1, j]) \end{pmatrix} si\ \ i>0\ \ et\ \ j>0 \quad (1.21)$$

$$E(i,j) = \begin{cases} max(E(i-1,j) - ge, S(i-1,j) - go) & si\ \ i>0\ \ et\ \ j>0 \\ S(0,j) - go & si\ \ i=0\ \ et\ \ j>0 \end{cases} \quad (1.22)$$

$$F(i,j) = \begin{cases} max(F(i,j-1) - ge, S(i,j-1) - go) & si\ \ i>0\ \ et\ \ j>0 \\ S(i,0) - go & si\ \ i>0\ \ et\ \ j=0 \end{cases} \quad (1.23)$$

Dans l'équation 1.22, lorque $i = 0$, on calcule $S(0,j) - go$. Cette opération ne sert qu'à initialiser la valeur de $E(0,j)$ à une valeur inférieure à celle de $S(0,j)$, afin de toujours commencer un alignement par une substitution (Cf. section 1.2.2). Il en est de même pour l'équation 1.23.

**Complexité de l'algorithme:** La récurrence complète a une complexité de $O(mn \times (m+n))$ puisque le calcul de $H(i,j)$ nécessite $O(max(m,n))$ opérations.

L'algorithme peut être amélioré en temps de calcul. En effet en prenant en compte quelques propriétés de la fonction $c$, la matrice $H$ peut être calculée en $O(m \times n)$. Définissons la matrice $L = (L(i,j))_{\substack{1 \le i \le n \\ 1 \le j \le m}}$ :

$$L(i,j) = \begin{cases} \min_{a_{i-t+1} = b_{j-t+1}} (t) & si\ \ t\ \ existe \\ l + 1 + min(i,j) & sinon. \end{cases} \quad (1.24)$$

$L(i,j)$ est égal à la distance au plus proche appariement exact s'il existe, et vaut $l + 1 + min(i,j)$ sinon. Définissons une autre matrice $D$, qui pourra être utilisée pour le calcul de $H$.

$$\begin{aligned} si\ \ L(i,j) &\le min(i,j)\ \ : \\ D(i,j) &= \max_{L(i,j) \le k \le min(i,j)} \Big( S(i-k, j-k) + c(A[i-k+1, i], B[j-k+1, j]) \Big) \\ sinon & \\ D(i,j) &= S\Big(i - min(i,j), j - min(i,j)\Big) + \\ & \quad c\Big(A[i - min(i,j) + 1, i], B[j - min(i,j) + 1, j]\Big) \end{aligned}$$

$D(i,j)$ est le score maximum pour aligner $A[1,i]$ et $B[1,j]$ qui finit par un bloc de substitutions contenant au moins un *match* si celui-ci existe, sinon, $D(i,j)$ est le score de l'alignement de $A[1,i]$ et $B[1,j]$ finissant par le plus long bloc de substitutions.
Huang [71] a montré que:

- pour $i > 0$ et $j > 0$, $H(i,j) = max\Big(S(i-1, j-1) + \sigma(a_i, b_j), D(i,j)\Big)$.

- si on pose $l_1 = L(i-1, j-1)$, on a :

$$D(i,j) = \begin{cases} S(i,j) & si \quad i=0 \quad ou \quad j=0 \\ D(i-1,j-1)+\sigma(a_i,b_j) & si \quad i,j>0 \quad et \quad a_i \neq b_j \\ S(i-1,j-1)+\sigma(a_i,b_j) & si \quad i,j>0 \quad et \quad a_i = b_j \quad et \quad l_1 > l \\ max\begin{pmatrix} S(i-1,j-1), \\ D(i-1,j-1)+ \\ \delta(a_{i-l_1}, b_{j-l_1})+ \\ \delta(a_i,b_j) \end{pmatrix} + \sigma(a_i,b_j) & si \quad i,j>0 \quad et \quad a_i = b_j \quad et \quad l_1 \leq l \end{cases}$$

En combinant les deux remarques précédentes, on obtient une récurrence pour le calcul de $S(i,j)$ :

$$S(i,j) = \begin{cases} max\begin{pmatrix} S(i-1,j-1)+\sigma(a_i,b_j), \\ D(i,j), E(i,j), F(i,j) \end{pmatrix} & si \quad i>0 \quad et \quad j>0 \\ -(go+ge.(i-1)) & si \quad i>0 \quad et \quad j=0 \\ -(go+ge.(j-1)) & si \quad i=0 \quad et \quad j>0 \\ 0 & si \quad i=0 \quad et \quad j=0 \end{cases} \tag{1.25}$$

$$D(i,j) = \begin{cases} S(i,j) & si \quad i=0 \quad ou \quad j=0 \\ D(i-1,j-1)+\sigma(a_i,b_j) & si \quad i,j>0 \quad et \quad a_i \neq b_j \\ S(i-1,j-1)+\sigma(a_i,b_j) & si \quad i,j>0 \quad et \quad a_i = b_j \quad et \quad l_1 > l \\ max\begin{pmatrix} S(i-1,j-1), \\ D(i-1,j-1)+ \\ \delta(a_{i-l_1}, b_{j-l_1})+ \\ \delta(a_i,b_j) \end{pmatrix} + \sigma(a_i,b_j) & si \quad i,j>0 \quad et \quad a_i = b_j \quad et \quad l_1 \leq l \end{cases} \tag{1.26}$$

$$E(i,j) = \begin{cases} max(E(i-1,j)-ge, S(i-1,j)-go) & si \quad i>0 \quad et \quad j>0 \\ S(0,j)-go & si \quad i=0 \quad et \quad j>0 \end{cases} \tag{1.27}$$

$$F(i,j) = \begin{cases} max(F(i,j-1)-ge, S(i,j-1)-go) & si \quad i>0 \quad et \quad j>0 \\ S(i,0)-go & si \quad i>0 \quad et \quad j=0 \end{cases} \tag{1.28}$$

$$L(i,j) = \begin{cases} L(i-1,j-1)+1 & si \quad i,j>0 \quad et \quad a_i \neq b_j \\ 1 & si \quad i,j>0 \quad et \quad a_i = b_j \\ l+1 & si \quad i=0 \quad ou \quad j=0 \end{cases} \tag{1.29}$$

**Alignement local :**

Pour adapter l'algorithme de SW, il suffit de modifier le calcul de $S(i,j)$, les autres équations restant inchangées. L'équation 1.25 est remplacée par :

$$S(i,j) = \begin{cases} max\begin{pmatrix} S(i-1,j-1)+\sigma(a_i,b_j), \\ D(i,j), E(i,j), F(i,j), 0 \end{pmatrix} & si \quad i>0 \quad et \quad j>0 \\ 0 & si \quad i=0 \quad ou \quad j=0 \end{cases} \tag{1.30}$$

### 1.5.5 Les algorithmes de programmation dynamique et leurs complexités

Nous n'avons survolé qu'une petite partie de la grande diversité des variantes des algorithmes de Needleman & Wunsch et de Smith & Waterman. Cependant pour avoir une plus large vue des

| Alignement | | Pénalité d'un gap | Complexité en | | Références. |
|---|---|---|---|---|---|
| Similarité | S[a]/A[b] | | temps | espace | |
| Globale | A | Pénalité / gap | $O(N^3)$ | $O(N^2)$ | [101] |
| Globale | A | Pénalité / résidu | $O(N^2)$ | $O(N^2)$ | [111] |
| Globale | S | Pénalité / résidu | $O(N^2)$ | $O(N)$ | [66] |
| Globale | S | Pénalité / résidu | $O(ND)$ | $O(ND)$ | [42] [126] |
| Globale | A | Arbitraire | $O(N^3)$ | $O(N^2)$ | [130] |
| Globale | S | $go + k.ge$ | $O(N^2)$ | $O(N)$ | [53] |
| | A | $go + k.ge$ | $O(N^2)$ | $O(N^2)$ | [92, 95] |
| Globale | A | $go + k.ge$ | $O(N^2)$ | $O(N)$ | [93] |
| Globale | S | $go + k.ge$ | $O(N^2)$ | $O(N)$ | [112] |
| | | Insertions/délétions de bords non pondérées sur une des 2 séquences | | | |
| Locale | A | $go + k.ge$ | $O(N^2)$ | $O(N^2)$ | [117] |
| K locales | A | $go + k.ge$ | $O(KN^2)$ | $O(N^2)$ | [132] |
| K locales | S | $go + k.ge$ | $O(KN^2)$ | $O(N)$ | [94] [95] |
| Locale | | Concave | $O(N^2 \log N)$ | | [92] |
| | | affine par morceau | $O(N^2 \log p)$ [c] | | [92] |
| | | Convexe | $O(N^2 \log N)$ | | [46] |
| | | CZP [d] | $O(N^2)$ | | [46] |
| Locale | S | $go + k.ge$ | $O(N^2)$ [e] | $O(N)$ | [54] |
| Globale avec profile | S | Profile ($go$ et $ge$ dépendent de la position) | $O(NM)$ | $O(N)$ | [55] [56] |
| Globale avec pattern | | Pattern | $O(MNlogN)$ | $O(N)$ | [16] [17] |
| Locale | | Restriction map comparison | | | [70] |
| Locale | A | Context dependant method | $O(N^2)$ | $O(N)$ | [71] |
| Locale | A | Pénalité dépendant de la position | $O(N^2)$ | $O(N)$ | [125] |

TAB. 1.3 - **Algorithmes de programmation dynamique et leurs complexités**

[a] S : seul le score est calculé

[b] A : l'alignement est donné en plus du score

[c] p est le nombre d'intervalles définissant la fonction de pénalité affine par morceau

[d] Closest Zero Property [46]

[e] L'algorithme de Green calcule le score SW si celui-ci est supérieur à $gap(1)$. La complexité au pire est $O(N^2)$, mais la complexité moyenne est inférieure. L'implémentation de Green est 25% à 30% plus rapide que l'implémentation habituelle!

problèmes et algorithmes liés à la programmation dynamique pour l'alignement de séquences, on pourra se référer aux articles cités dans le tableau 1.3.

## 1.6 Algorithme générique de la programmation dynamique pour l'alignement de séquences

Les algorithmes de programmation dynamique sont largement répandus dans des domaines très différents, comme des problèmes de recherche de plus court chemin dans un graphe, ou comme des problèmes d'optimisation (on peut penser à la phase d'apprentissage d'une chaîne de Markov à états cachés [107]). Nous avons déjà abordé les liens entre l'alignement de séquences et le problème de recherche de plus court chemin dans un graphe orienté. Ici, nous montrons que le même algorithme peut conduire à la résolution de problèmes différents sur les séquences.

**L'algorithme générique :** On se donne deux séquences $A$ et $B$ de longueur respective $n$ et $m$. Pour chaque double indice $(i,j), i \in [1;n] \quad et \quad j \in [1;m]$, on calcule la valeur $M[i,j]$ donnée par la récurrence suivante :

$$\begin{aligned}
M[0,0] &= init_0 \\
M[i,0] &= f\Big(M[i-1,0] \ , \ c(A_i,_)\Big) \\
M[0,j] &= f\Big(M[0,i-1] \ , \ c(_,B_i)\Big) \\
M[i,j] &= g\left(\begin{array}{lll} f( & M[i-1,j-1] & , \ c(A_i,B_j) \ ) \\ f( & M[i-1,j] & , \ c(A_i,-) \ ) \\ f( & M[i,j-1] & , \ c(-,B_j) \ ) \end{array}\right)
\end{aligned}$$

Le tableau 1.4 explique comment cet algorithme générique répond aux problèmes suivants :

- La recherche de la plus longue sous-séquence commune,
- La recherche du score NW ou SW,
- Le calcul de la probabilité d'un alignement, suivant un certain modèle : on se donne les probabilités d'observer dans un alignement deux lettres identiques en face ($P(match)$), d'observer deux lettres différentes alignées ($P(mismatch)$) et d'observer une insertion/délétion ($P(insert)$ et $P(delete)$). L'algorithme de programmation dynamique trouve l'alignement le plus probable pour deux séquences. Malheureusement, ces probabilités deviennent vraiment très petites, et entraînent un *underflow*. Il est alors intéressant de prendre les logarithmes des probabilités, ce qui correspond à une interprétation du type théorie de l'information ou du codage.
- Le calcul de la longueur du message minimum [9] [3, 4] : on cherche la plus courte façon de transmettre les deux séquences à aligner à travers un système d'émetteur-transmetteur. Si les deux séquences n'ont pas de similarité, la longueur du message sera proche de la somme des deux longueurs des séquences. Par contre si les deux séquences sont similaires, le message pourra être plus court. Et le meilleur alignement mènera au plus court message.

9. On parle souvent du problème *MML* qui est l'acronyme anglais de «Minimum Message Length».

On se donne alors un modèle a priori pour un alignement, considéré comme une chaîne de caractères sur l'alphabet des paires de lettres (le caractère «−» étant pris en compte). Chaque événement reçoit une probabilité.

Si l'alignement génère une compression de l'information, alors, l'hypothèse que les deux séquences soient reliées est acceptée.

Notons que le problème de la longueur du message minimum est strictement équivalent à celui du calcul de la probabilité d'un alignement. Le passage de la probabilité à la longueur du message minimum se fait en prenant les logarithmes des probabilités, et, au lieu de chercher le maximum, on est amené à minimiser le logarithme.

| méthodes | $init_0$ | $g(\ )$ | $f(\ )$ | $c(\ )$ |
|---|---|---|---|---|
| LCS[a] | 0 | max( ) | + | $c(x,x)=1$<br>$c(x,y)=c(x,-)=c(-,y)=0$ |
| NW | 0 | $\max(x,y,z)$ | + | $c(x,y)=\sigma(x,y)$<br>$c(x,-)=c(-,y)=-\delta$ |
| SW | 0 | $\max(x,y,z,0)$ | + | $c(x,y)=\sigma(x,y)$<br>$c(x,-)=c(-,y)=-\delta$ |
| Probabilité | 1 | max( ) | × | $c(x,x)=P(match)\times P(x)$<br>$c(x,y)=P(mismatch)\times P(x,y\|x\neq y)$<br>$c(x,-)=P(delete)\times P(x)$<br>$c(-,x)=P(insert)\times P(x)$ |
| MML[b] | 0 | min( ) | + | $c(x,x)=-\log_2(P(match))-\log_2(P(x))$<br>$c(x,y)=-\log_2(P(mismatch))-\log_2(P(x,y\|x\neq y)$<br>$c(x,-)=-\log_2(P(delete))-\log_2(P(x))$<br>$c(-,x)=-\log_2(P(insert))-\log_2(P(x))$ |

TAB. 1.4 - **Les divers algorithmes de programmation dynamique pour les séquences biologiques**

[a] Longest Common Subsequence
[b] Minimum Message Length [3]

## 1.7 Théorie générale de la programmation dynamique

La programmation dynamique est une méthode générale, pour résoudre des problèmes lorsque des décisions sont prises successivement. Le résultat de la décision n'est pas toujours prévisible. Dans le cas déterministe, le résultat ne dépend que de la décision, alors que dans le cas stochastique, le hasard s'en mêle. Cependant dans les deux cas, le résultat est observable, avant la décision suivante. L'objectif est de minimiser (resp. *maximiser*) un coût, au cours des différentes décisions.

Une très grande classe de problèmes peut être traitée de cette manière. Même si cette méthode est beaucoup plus générale, on se limitera à une présentation dans le cas des systèmes dynamiques

à temps discret et d'une fonction de coût additive. Le système dynamique est de la forme :

$$x_{k+1} = f_k(x_k, u_k, w_k) \quad k = 0, 1, \ldots, N-1$$

où

- $k$ est le temps,
- $x_k$ est l'état du système au temps $k$,
- $u_k$ est la variable modélisant la décision prise au temps $k$,
- $w_k$ est un paramètre aléatoire,
- $N$ est l'horizon du problème.

La fonction de coût est additive : elle est égale à la somme de chacun des coûts élémentaires associés à chacune des décisions :

$$\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k).$$

Puisque dans le cas général, à cause de la présence de $w_k$, le coût est une fonction aléatoire, nous cherchons à optimiser l'espérance du coût global :

$$E\Big(\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\Big).$$

Les variables $x_k$, $u_k$ et $w_k$ sont prises dans des ensembles $X_k$, $U_k$ et $W_k$. La commande $u_k$ est à prendre parmi un sous-ensemble $U_k(x_k)$ de $U_k$ qui ne dépend que de $x_k$ et $k$. La variable aléatoire $w_k$ est caractérisée par la distribution $P(.|x_k, u_k)$ qui ne dépend que du «présent».

Etant donné un état initial $x_0$, le problème est de trouver une suite de commande $U = (u_0, u_1, \ldots)$ qui minimise la fonctionnelle :

$$J_U(x_0) = E\Big(\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\Big)$$

On appelle *politique optimale*, toute suite de commandes $U^*$ qui minimise le coût global : $J_{U^*}(x_0) = \min_U J_U(x_0)$.

La technique de la programmation dynamique repose sur ***le principe d'optimalité***. Le nom est dû à Bellman [19], qui a contribué au développement et à la diffusion de cet outil. Soit $U^* = (u_0^*, u_1^*, \ldots)$ une politique optimale. Considérons le sous-problème où nous sommes au temps $i$ à l'état $x_i$. $x_i$ est l'état du système au temps $i$, lorsque les décisions $(u_0^*, u_1^*, \ldots, u_{i-1}^*)$ sont celles de la politique optimale. Nous voulons optimiser le coût à partir du temps $i$ :

$$E\Big(\sum_{k=i}^{N-1} g_k(x_k, u_k, w_k)\Big)$$

Alors, la politique optimale tronquée $(u_i^*, u_{i+1}^*, \ldots, u_{N-1}^*)$ est optimale pour ce sous-problème.

La justification *intuitive* de ce principe d'optimalité est simple. Si la politique optimale tronquée $(u_i^*, u_{i+1}^*, \ldots, u_{N-1}^*)$ n'était pas optimale pour le sous-problème, nous serions capables dans le cas du

problème non tronqué, de réduire le coût global en remplaçant $(u_i^*, u_{i+1}^*, \ldots, u_{N-1}^*)$ de la politique initiale par une politique optimale pour le sous-problème.

Si nous nous plaçons dans le cas déterministe, l'équation d'état devient $x_{k+1} = f_k\Big(x_k, u_k\Big)$. Le coût total sur l'ensemble des décisions entre l'état initial $x_0$ au temps 0, et l'état final au temps $N$ est donné par:

$$C_{0,x_0}(u) = \sum_{t=0}^{N-1} g_k\Big(x_k, u_k\Big). \tag{1.31}$$

On plonge alors ce problème, noté $(\mathcal{P}_{0,x=x_0})$, dans une famille de problèmes de même nature $(\mathcal{P}_{k,x=x_k})$:

trouver la suite $(u)$ de commandes qui minimise le coût global:

$$V(x,k) = \min_u C_{k,x}(u) = \min_u \sum_{s=k}^{N-1} g_s\Big(x_s, u_s\Big) \tag{1.32}$$

Le principe de Bellman consiste à décomposer la suite de commandes $(u_0^*, u_1^*, \ldots)$ en une première commande $u_0^*$ et une sous-suite $(u_1^*, u_2^*, \ldots)$, puis à dire que toute solution optimale doit maximiser la somme du coût de la commande $u_0^*$ et du coût optimal pour $(u_1^*, u_2^*, \ldots)$. On découpe le problème initial en sous-problèmes de compléxités inférieures. On peut formaliser ce principe:

**Principe de Bellman** [19]:

$$\begin{aligned} V(x_k,k) &= \min_u \sum_{s=k}^{N-1} g_s\Big(x_s, u_s\Big) \\ &= \min_{u_k}\Big\{ g_k\Big(x_k, u_k\Big) + V\Big(\underbrace{f_k\Big(x_k, u_k\Big)}_{=x_{k+1}}, k+1\Big)\Big\} \\ avec \qquad & V(x_N, N) = 0 \end{aligned} \tag{1.33}$$

Si on reprend la description de l'algorithme de Needleman & Wunsch, le principe de Bellman n'est pas explicite. Le vocabulaire utilisé pour l'alignement de séquences diffère de celui de Bellman. Le tableau 1.5 donne la correspondance entre le principe de Bellman et l'algorithme de Needleman & Wunsch. Que ce soit dant le cas des pénalités linéaires ou affines, les états peuvent être vus comme les sommets du graphe orienté valué associé (Cf. figures 1.1 et 1.6). Les commandes sont équivalentes aux arcs de ces graphes, et les coûts élémentaires aux poids des arcs.

**Remarque:** L'algorithme de Smith & Waterman ne correspond pas à une situation classique de la programmation dynamique. Il s'agit de la recherche d'une sous-trajectoire optimale. En reprenant le formalisme de la programmation dynamique, l'algorithme SW s'exprime par:

$$SW = \max_{u,k_1,k_2} \sum_{k=k_1}^{k_2} g_k(x_k, u_k).$$

L'optimisation est faite non seulement sur la suite de commandes mais aussi sur les bornes de l'intervalle de temps $[k_1, k_2]$.

| **Programmation dynamique** | | **Comparaison de séquences** |
|---|---|---|
| Etat | $x_k$ | position dans le graphe<br>$(i,j)_D$, $(i,j)_H$, $(i,j)_V$ |
| Commande | $u_k$ | transition<br>$(i,j)_D \longrightarrow (i+1,j+1)_D$<br>$(i,j)_D \longrightarrow (i+1,j)_V$<br>$(i,j)_D \longrightarrow (i,j+1)_H$<br>$(i,j)_V \longrightarrow (i,j)_D$<br>$(i,j)_V \longrightarrow (i+1,j)_V$<br>$(i,j)_H \longrightarrow (i,j)_D$<br>$(i,j)_H \longrightarrow (i,j+1)_H$ |
| Temps | $k$ | longueur du chemin |
| Coût élémentaire | $g_k$ | $\sigma_{i,j}$, $go$, $ge$ |
| Coût optimal partiel | $V(x,k)$ | $NW_{i,j}$ |

TAB. 1.5 - **Correspondance entre la programmation dynamique et l'alignement de séquences** *par l'algorithme Needleman & Wunsch. Les coûts de pénalisation des insertions/délétions sont supposés affines :* $g(k) = go + (k-1) \times ge$. *Dans le cas d'une fonction de pénalisation linéaire, il n'y a qu'un état par couple* $(i,j)$. *Les états et les transitions sont donnés par les sommets et les arcs du graphe associé (Cf. figures 1.1 et 1.6).*

# Chapitre 2

# Autres regards sur la programmation dynamique en bio-informatique

*La première limitation de l'algorithme de Smith & Waterman est celle du temps d'exécution. Cet algorithme a une complexité quadratique. Puisque les banques de données croissent de manière exponentielle, le temps nécessaire à la comparaison d'une séquence à toutes celles d'une banque devient rapidement difficile à gérer. La classification de séquences est encore beaucoup plus gourmande en temps, puisqu'elle nécessite la comparaison de toutes les séquences d'une banque de données entre elles. Quelques améliorations peuvent être apportées aux algorithmes de programmation dynamique. Dans le cas de l'alignement global, le simple fait de reformuler la programmation dynamique permet de diminuer le nombre d'opérations au sein de la boucle de l'algorithme.*

*Dans le cas de l'alignement local (c'est aussi vrai dans le cas de l'alignement global), l'utilisation des instructions dédiées à la gestion du graphique, permet de traiter plusieurs colonnes de la matrice de scores en même temps. Mais cela nécessite de borner les différences entre les valeurs de la matrice de scores.*

*Nous nous focaliserons ensuite sur les relations entre les algorithmes de Needleman & Wunsch et de Smith & Waterman : l'alignement local peut être vu comme une optimisation de l'alignement global sur l'ensemble des sous-séquences. D'autre part, on exhibe une expression exacte du nombre d'alignements possibles entre deux séquences de longueurs données.*

*Pour l'alignement lui-même, et non plus seulement le score, la phase de remontée est déterminante. Suivant l'implémentation de la remontée, différentes régions de similarité peuvent être soulignées. Les alignements trouvés peuvent alors correspondre à ceux trouvés par d'autres alogrithmes plus complexes.*

## 2.1 Une nouvelle formulation pour Needleman & Wunsch : un léger gain

Lorsqu'on reprend l'équation de Bellman 1.33, on est tenté, dans le cas de l'algorithme de Needleman & Wunsch, de modifier la fonction de coût total : on cherche à exprimer que le score final se lit à la dernière itération de calcul, correspondant aux indices $(n, m)$ des longueurs des deux séquences considérées. On peut alors prévoir d'aller par des verticales ou des horizontales jusqu'à la dernière case. La valeur modifiée du tableau $NW_m$ en $(i, j)$ correspond alors au coût du chemin de $(0, 0)$ à $(n, m)$ passant par la case $(i, j)$ et finissant uniquement par des insertions/délétions.

Etudions tout d'abord le cas simplifié où la pénalité des insertions/délétions est linéaire.

### 2.1.1 Pénalité linéaire des insertions/délétions : $gapo = gape = \delta$

Soient deux séquences $A$ et $B$ de longueur respective $n$ et $m$. Soit $NW_{i,j}$ le score d'alignement Needleman & Wunsch entre les séquences $A[1,n]$ et $B[1,m]$, en imposant que l'alignement se termine par au moins $m-j$ insertions dans $B$ (Cf. figure 2.1).

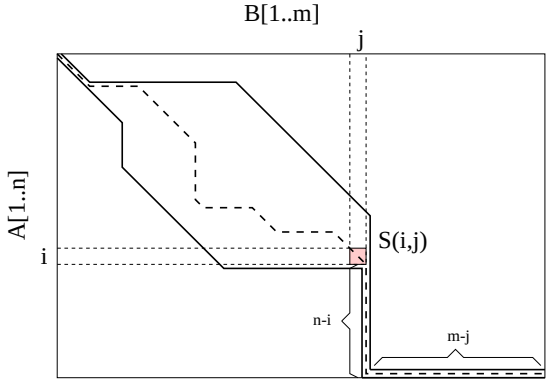



FIG. 2.1 - **Amélioration** *de l'algorithme de Needleman & Wunsch. On modifie la définition de la variable* $NW_{i,j}$.

On a les relations de récurrence suivantes :

$$NW_{i,j} = \max \begin{pmatrix} NW_{i-1,j}, \\ NW_{i-1,j-1} + \sigma_{i,j} + \delta + \delta, \\ NW_{i,j-1} \end{pmatrix} \tag{2.1}$$

où $\sigma_{i,j}$ est la valeur de la matrice de substitution pour les lettres $A[i]$ et $B[j]$. Les initialisations sont données par :

$$NW_{i,0} = (n+m) \times \delta \quad \forall i \in [0,n] \tag{2.2}$$
$$NW_{0,j} = (n+m) \times \delta \quad \forall j \in [0,m] \tag{2.3}$$

Comme dans la formulation habituelle, on a, à chaque étape, 3 sommes, et un max entre trois valeurs. Et il est nécessaire de retenir un vecteur complet $NW_{*,j}$.
Mais $\sigma_{i,j}$ apparaît toujours avec $2 \times \delta$. Si on construit au préalable la matrice de similarité modifiée ($\sigma'_{i,j} = \sigma_{i,j} + 2\delta$), on obtient les relations de récurrence :

$$NW_{i,j} = \max \begin{pmatrix} NW_{i,j-1}, \\ NW_{i-1,j-1} + \sigma'_{i,j}, \\ NW_{i-1,j} \end{pmatrix} \tag{2.4}$$

| opérations | algorithme habituel | nouvelle implémentation | **gain** |
|---|---|---|---|
| addition | 3 | 1 | 2 |
| max | 2 | 2 | 0 |

Cette modification amène un **gain de 2 additions pour chaque couple d'indices** $(i,j)$.

### 2.1.2 Pénalité affine des insertions/délétions : $gapo \neq gape$

Soit $g$ la fonction de pénalisation des insertions/délétions : $g(k) = go + (k-1) \times ge$. Définissons les variables suivantes :

- $D_{i,j}$ : score d'alignement de $A[1,n]$ et de $B[1,m]$ en mettant en correspondance les lettres $A[i]$ et $B[j]$ et se terminant par $n-i$ délétions puis $m-j$ insertions. L'alignement se termine par :

$$\begin{array}{ccc} A_i & A[i+1,n] & - \ - \ - \\ B_j & - \ - \ - & B[j+1,m] \end{array}$$

- $V_{i,j}$ : score d'alignement de $A[1,n]$ et de $B[1,m]$ en sachant que l'alignement se termine par la délétion des $n-i+1$ lettres ($A[i,n]$) et par $m-j$ insertions. La fin de l'alignement est :

$$\begin{array}{ccc} A_i & A[i+1,n] & - \ - \ - \\ - & - \ - \ - & B[j+1,m] \end{array}$$

- $H_{i,j}$ : score d'alignement de $A[1,n]$ et de $B[1,m]$ en sachant que l'alignement se termine par :

$$\begin{array}{ccc} - & A[i+1,n] & - \ - \ - \\ B_j & - \ - \ - & B[j+1,m] \end{array}$$

Les relations de récurrence deviennent (à l'intérieur du tableau) :

$$D_{i,j} = \max \begin{pmatrix} D_{i-1,j-1}, \\ V_{i-1,j-1} - go + ge, \\ H_{i-1,j-1} \end{pmatrix} + 2.ge + \sigma_{i,j} \tag{2.5}$$

$$V_{i,j} = \max \begin{pmatrix} D_{i-1,j}, \\ V_{i-1,j} \end{pmatrix} \tag{2.6}$$

$$H_{i,j} = \max \begin{pmatrix} H_{i,j-1}, \\ D_{i,j-1} - go + ge \end{pmatrix} \tag{2.7}$$

Si on construit la matrice de similarité modifiée ($\sigma'_{i,j} = \sigma_{i,j} + 2.ge$), on observe un **gain de 2 additions pour chaque couple d'indices (i ,j)**.

| opérations | algorithme habituel | nouvelle implémentation | **gain** |
|---|---|---|---|
| addition | 5 | 3 | 2 |
| max | 4 | 4 | 0 |

Les équations précédentes sont valables uniquement dans le creux du tableau. En effet, Lorsqu'on est sur la dernière ligne, le calcul de $D_{n,j}$ doit prendre en compte le fait que la verticale pour rejoindre la dernière ligne disparaît. La récurrence devient :

$$D_{i,j} = \max \begin{pmatrix} D_{i-1,j-1}, \\ V_{i-1,j-1} - go + ge, \\ H_{i-1,j-1} \end{pmatrix} + go + ge + \sigma_{i,j} \tag{2.8}$$

$$V_{i,j} = \max \begin{pmatrix} D_{i-1,j}, \\ V_{i-1,j} \end{pmatrix} \tag{2.9}$$

$$H_{i,j} = \max \begin{pmatrix} H_{i,j-1}, \\ D_{i,j-1} + ge \end{pmatrix} \tag{2.10}$$

La même remarque est à faire pour la dernière colonne. On doit tenir compte que la dernière horizontale pour rejoindre la dernière case disparaît.

Il est à remarquer que cette modification amène à conserver une variable supplémentaire : $V_{i,j}$, puisque cette variable se distingue dans la récurrence de la programmation dynamique. En effet, pour le calcul de $D_{i,j}$ (éq. 2.5), il faut distinguer deux cas :

- soit on provient de la verticale,
- soit on provient d'une diagonale ou d'une horizontale.

Dans le cas où le chemin provient de la verticale, le nombre total d'insertions/délétions a été augmenté de 1, et il faut retrancher une pénalité d'ouverture d'insertion/délétion *gapo*. Dans l'autre cas, le nombre total de gaps reste inchangé (Cf. figure 2.2).

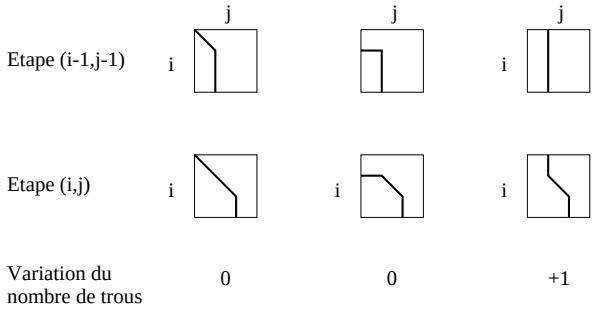



FIG. 2.2 - **Amélioration** *de l'algorithme de Needleman & Wunsch. Une variable supplémentaire doit être conservée. Pour le calcul de $D_{i,j}$ (éq. 2.5), le nombre d'ouvertures d'insertion/délétion change suivant qu'on provient de la verticale ou non.*

Les deux figures suivantes (2.3 et 2.4) montrent les flux de données au sein de la boucle de l'algorithme Needleman & Wunsch et de l'algorithme modifié. Les flux s'enchevêtrent moins dans le cas de l'algorithme modifié, les dépendances ont été diminuées. La boucle de l'algorithme a été simplifiée.

Puisque le flux des données à l'intérieur de l'algorithme est plus simple, on peut envisager une parallélisation de l'algorithme à un bas niveau. Les opérations élémentaires peuvent se faire deux par deux : d'abord 2 additions, puis deux opérations de maximisation, enfin les deux dernières opérations de maximisation et d'addition.

## 2.2 Parallélisation de SW par instruction vidéo et relation entre les valeurs de la matrice de scores

L'utilisation des instructions vidéo dans l'algorithme de Smith & Waterman rend possible une parallélisation de bas niveau : 2, 4, ou 8 instructions peuvent être effectuées en même temps. Il existe des implémentations parallèles de cet algorithme sur des cartes spécialisées [32][85] mais ces implémentations sont chères, difficilement programmables, et les cartes ne peuvent pas être réutilisées pour d'autres besoins. L'implémentation purement *software* sur des plateformes multiprocesseurs [48], est une alternative intéressante. L'implémentation en parallèle au niveau des instructions est due à A. Wozniak [136], mais la preuve de la validité de l'algorithme nécessite de

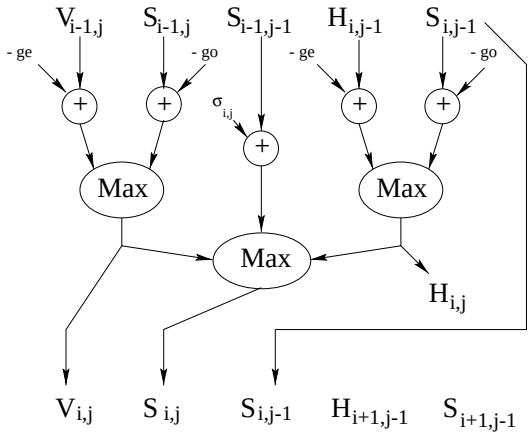



FIG. 2.3 - **Une étape de l'algorithme Needleman & Wunsch** *dans l'implémentation habituelle. Pour chaque étape, il y a* 5 *additions et* 4 *max.*

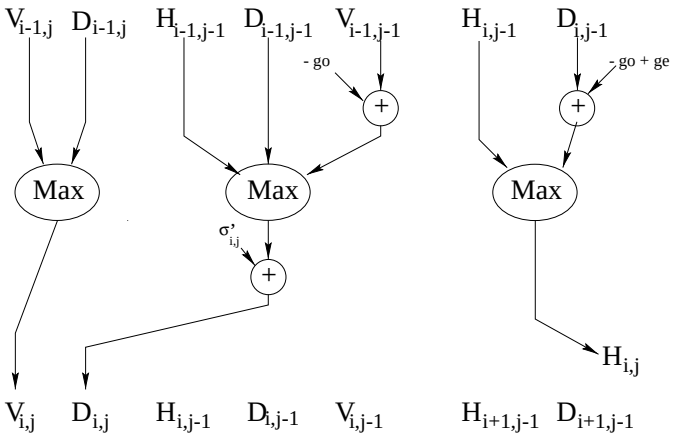



FIG. 2.4 - **Une étape de l'algorithme Needleman & Wunsch modifié.** *Pour chaque étape, il y a* 3 *additions et* 4 *max.*

borner les différences entre les valeurs obtenues à l'intérieur de la boucle principale de la programmation dynamique.

Si on choisit correctement quatre positions $(i, j)$ dans la matrice de scores, on peut mener les quatre calculs de manière parallèle : les résultats obtenus sur l'une d'entre elles n'influe pas sur le calcul des autres (Cf. figure 2.5). Si les valeurs ne sont pas trop grandes, les quatre calculs peuvent être effectués simultanément sur un seul processeur.

### 2.2.1 Positionnement du problème

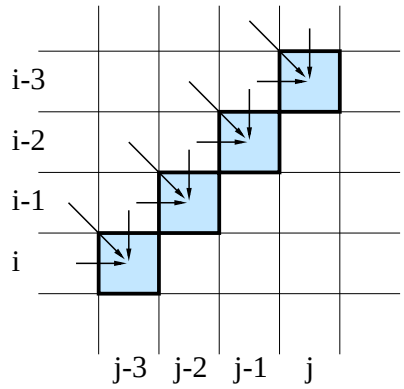



FIG. 2.5 - **Calcul du score Smith & Waterman** *sur 4 cellules en même temps. Les calculs sur les quatre cellules ombrées sont indépendants et peuvent être effectués simultanément.*

Les instructions VIS[1] utilisent des registres de 64 bits, et permettent des opérations en parallèle soit sur deux entiers de 32 bits, soit sur 4 entiers de 16 bits, codés en un seul mot de 64 bits. Les opérations existantes sont les additions, soustractions, comparaisons, mais il n'y a pas l'opération max qui nous intéresse. L'opération max sur 2 entiers de 32 bits chacun sans utiliser d'instructions de branchement, s'écrit comme suit :

```
#define MAX(int1, int2)\
{\
        int mask;\
        int1 -= int2;\
        mask = (int1>>(8*sizeof(int1)-1));\
        int1 &= ~mask;\
        int1 += int2;\
}
```

*int1* et *int2* contiennent les deux valeurs initiales dont on cherche le maximum. Le résulat est stocké dans *int1*. On calcule d'abord la différence.

- Si cette différence est positive, alors le maximum est la première valeur (le *mask* ne contient que des 0).
- Si la différence est négative, le *mask* ne contient que des 1, et *int1* prend la valeur *int2*.

1. Visual Instruction Set

Une suite d'instructions du même type peut être écrite, pour calculer sans branchement et simultanément $max(i_1, i_2)$ et $max(j_1, j_2)$, lorsque $i_1$ et $j_1$ sont concaténés dans la même entité mémoire de 32 bits et que $i_2$ et $j_2$ le sont aussi.

Le résultat de l'algorithme n'est correct que si l'on sait que le score ne dépasse pas 32767, valeur maximale d'un entier signé codé sur 16 bits. Le résultat est juste pour les valeurs inférieures. Lors des additions successives, il est facile de contrôler si l'une des valeurs dépasse le seuil de 32767. Par contre, au sein de cette fonction MAX, il peut y avoir saturation lors de l'opération de soustraction. Si $int1 - int2 \geq 32767$, la fonction MAX ne donne plus le bon résultat. On est donc amené à borner les différences entre les valeurs dont on veut extraire le maximum.

Nous montrons que les différences entre les valeurs voisines sont bornées, et donnons un majorant de cette différence. Ainsi, tant qu'on n'obtient pas de valeurs proches de 32767, on peut assurer la validité des résultats.

On rappelle la relation de récurrence :

$$SW_{i,j} = \max \begin{pmatrix} SW_{i-1,j-1} + S(A_i, B_j), \\ D_{i,j}, \\ I_{i,j}, \\ 0 \end{pmatrix} \tag{2.11}$$

Soit $g(.)$ la fonction affine de pénalisation des insertions/délétions. Suivant les auteurs, $g(1) = gapo$ ou $g(1) = gapo + gape$. En respectant nos conventions, on a $g(1) = gapo$.

Posons les notations suivantes :

$$\begin{aligned} S_{i,j} &= S(A_i, B_j) \\ M &= \max_{i,j} S_{i,j} \\ m &= \min_{i,j} S_{i,j} \\ \Delta_{i,j} &= SW_{i-1,j-1} + S_{i,j} \end{aligned}$$

L'introduction de la variable $\Delta_{i,j}$ est purement formelle, elle représente la valeur du meilleur chemin se terminant par un appariement des lettres $A_i$ et $B_j$ ( Diagonale ). Essayons de donner une majoration de l'écart entre $\Delta_{i,j}$ et $\Delta_{i-1,j}$.

### 2.2.2 Relation entre les valeurs $\Delta_{i,j}$ et $\Delta_{i-1,j}$

Considérons tout d'abord que $\Delta_{i,j} \leq \Delta_{i-1,j}$. Deux cas se présentent.

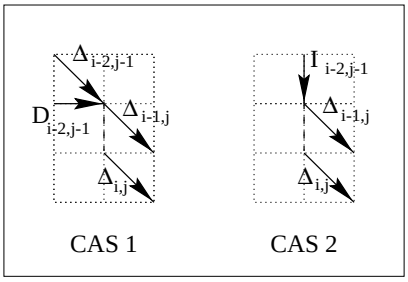



FIG. 2.6 - **Les deux cas possibles** *lorsque* $\Delta_{i,j} \leq \Delta_{i-1,j}$.

– Cas 1: Supposons que $\Delta_{i-1,j} = \Delta_{i-2,j-1} + S_{i-1,j}$, ou que $\Delta_{i-1,j} = D_{i-2,j-1} + S_{i-1,j}$ ( Cas 1, figure 2.6 ).

$$\Delta_{i,j} \geq \Delta_{i-1,j} - S_{i-1,j} - g(1) + S_{i,j}$$

– Cas 2: Supposons que $\Delta_{i-1,j} = I_{i-2,j-1} + S_{i-1,j}$ ( Cas 2, figure 2.6 ).

$$\Delta_{i,j} \geq \Delta_{i-1,j} - S_{i-1,j} - ge + S_{i,j}$$

Dans le cas où $\Delta_{i,j} \leq \Delta_{i-1,j}$, on peut majorer l'écart entre ces deux valeurs :

$$\Delta_{i-1,j} - \Delta_{i,j} \leq M - m + g(1) \tag{2.12}$$

Considérons maintenant que $\Delta_{i,j} \geq \Delta_{i-1,j}$. Six cas se présentent :

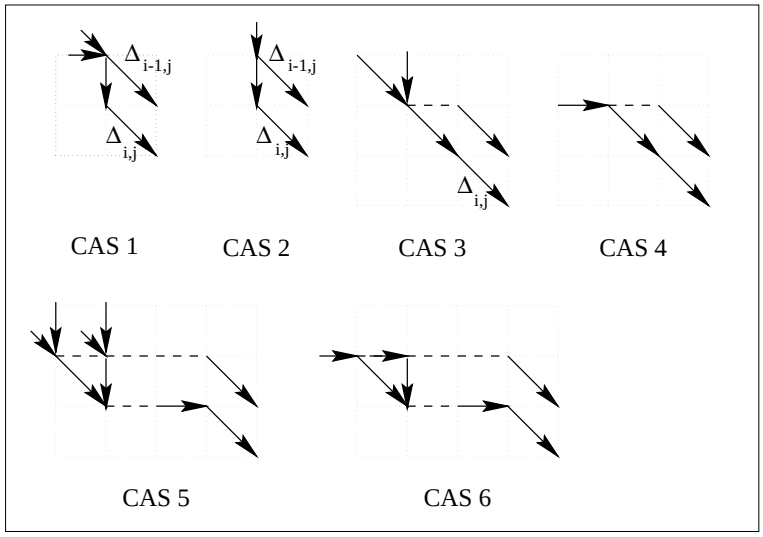



FIG. 2.7 - **Les six cas possibles** *lorsque* $\Delta_{i,j} \geq \Delta_{i-1,j}$.

– Cas 1: Supposons que $\Delta_{i,j} = I_{i-1,j-1} + S_{i,j}$ et que $I_{i-1,j-1} = \max(\Delta_{i-1,j-2}, D_{i-1,j-2}) - g(1)$ ( Cas 1, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} + g(1) + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq M - m - g(1) \end{aligned} \tag{2.13}$$

– Cas 2: Supposons que $\Delta_{i,j} = I_{i-1,j-1} + S_{i,j}$ et que $I_{i-1,j-1} = I_{i-1,j-2} - ge$ ( Cas 2, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} + ge + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq M - m - ge \end{aligned}$$

- Cas 3: Supposons que $\Delta_{i,j} = \Delta_{i-1,j-1} + S_{i,j}$ et que $\Delta_{i-1,j-1} = Max(I_{i-2,j-2}, \Delta_{i-2,j-2}) + S_{i-1,j-1}$ ( Cas 3, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} - S_{i-1,j-1} - g(1) + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq 2.M - m + g(1) \end{aligned}$$

- Cas 4: Supposons que $\Delta_{i,j} = \Delta_{i-1,j-1} + S_{i,j}$ et que $\Delta_{i-1,j-1} = D_{i-2,j-2} + S_{i-1,j-1}$ ( Cas 4, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} - S_{i-1,j-1} - ge + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq 2.M - m + ge \end{aligned}$$

- Cas 5 et 6: Supposons que $\Delta_{i,j} = D_{i-1,j-1} + S_{i,j}$. Alors, il existe $k \geq 0$ tel que $D_{i-1,j-1} = SW_{i-1,j-1-k} - g(k)$ avec $SW_{i-1,j-1-k} \neq D_{i-1,j-1-k}$. Quatre cas sont à distinguer:

  1. $SW_{i-1,j-1-k} = \max(\Delta_{i-2,j-2-k}, I_{i-2,j-2-k}) + S_{i-1,j-1-k}$ ( Cas 5, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} \\ &\quad -S_{i-1,j-1-k} - ge + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq 2.M - m + ge \end{aligned}$$

  2. $SW_{i-1,j-1-k} = D_{i-2,j-2-k} + S_{i-1,j-1-k}$ ( Cas 6, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} - S_{i-1,j-1-k} \\ &\quad -ge + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq 2.M - m + ge \end{aligned}$$

  3. $SW_{i-1,j-1-k} = I_{i-1,j-1-k} = I_{i-2,j-1-k} - ge$ ou $SW_{i-1,j-1-k} = I_{i-1,j-1-k} = \Delta_{i-2,j-1-k} - g(1)$ ( Cas 5, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} \\ &\quad +ge + S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq M - m - ge \end{aligned}$$

  4. $SW_{i-1,j-1-k} = I_{i-1,j-1-k} = D_{i-2,j-1-k} - g(1)$ ( Cas 6, figure 2.7 ).

$$\begin{aligned} \Delta_{i-1,j} &\geq \Delta_{i,j} - S_{i,j} + g(0) + g(1) \\ &\quad +S_{i-1,j} \\ \Delta_{i,j} - \Delta_{i-1,j} &\leq M - m - ge - 2.g(0) \end{aligned} \tag{2.14}$$

Si $\Delta_{i,j} \geq \Delta_{i-1,j}$, alors on peut majorer l'écart entre ces deux valeurs, en tenant compte que $g(1) \geq ge$. Les équations 2.13 à 2.14 peuvent se résumer en:

$$\Delta_{i,j} - \Delta_{i-1,j} \leq 2.M - m + g(1) \tag{2.15}$$

Dans tous les cas, on peut écrire en rassemblant les inéquations 2.12 et 2.15 que:

$$| \Delta_{i,j} - \Delta_{i-1,j} | \leq 2.M + g(1) - m \tag{2.16}$$

De la même manière, on peut montrer que l'écart entre $\Delta_{i,j}$ et $\Delta_{i,j-1}$ est majoré par le même majorant (Cf. éq. 2.16). En fait, on a prouvé que la différence en valeurs absolues $|\Delta_{i,j} - \Delta_{i-1,j}|$ est moins importante quand $\Delta_{i,j} \leq \Delta_{i-1,j}$ que lorsque $\Delta_{i,j} \geq \Delta_{i-1,j}$.

### 2.2.3 Relation entre les valeurs $\Delta_{i,j}$ et $D_{i,j}$

On rappelle la relation de récurrence :

$$D_{i,j} = \max\Big( SW_{i,j-1} - g(1) \ , \ D_{i,j-1} - ge\Big)$$

On a toujours l'inéquation suivante : $D_{i,j} \geq \Delta_{i,j-1} - g(1)$. En utilisant la relation 2.16, on obtient :

$$D_{i,j} \quad \geq \quad \Delta_{i,j} - (2.M + g(1) - m) - g(1)$$

Pour avoir un encadrement de la valeur de $D_{i,j}$, il faut distinguer plusieurs cas. Considérons tout d'abord que $D_{i,j} \leq \Delta_{i,j}$.

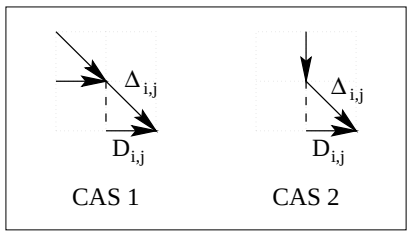



FIG. 2.8 - **Les deux cas possibles** *lorsque* $D_{i,j} \leq \Delta_{i,j}$

- Cas 1: $\Delta_{i,j} = \max(SW_{i-2,j-2} + S_{i-1,j-1}, D_{i-1,j-1}) + S_{i,j}$ ( Cas 1, figure 2.8 ).

$$D_{i,j} \quad \geq \quad \Delta_{i,j} - S_{i,j} - 2.g(1)$$

- Cas 2: $\Delta_{i,j} = D_{i-1,j-1} + S_{i,j}$ ( Cas 2, figure 2.8 ).

$$D_{i,j} \quad \geq \quad \Delta_{i,j} - S_{i,j} - ge - g(1)$$

On a alors, puisque $ge \leq g(1)$ :

$$\Delta_{i,j} - D_{i,j} \quad \leq \quad M + 2.g(1) \tag{2.17}$$

Considérons maintenant que $D_{i,j} \geq \Delta_{i,j}$. Alors, il existe $k \geq 1$ tel que $D_{i,j} = \max(SW_{i-1,j-k-1} + S_{i,j-k}, I_{i,j-k}) - g(k)$.

- supposons que $D_{i,j} = SW_{i-1,j-k-1} + S_{i,j-k} - g(k)$ ( Cas 1, figure 2.9 ).

$$\begin{aligned} \Delta_{i,j} \quad &\geq \quad D_{i,j} - S_{i,j-k} + S_{i,j} \\ D_{i,j} - \Delta_{i,j} \quad &\leq \quad M - m \end{aligned}$$

- Supposons que $D_{i,j} = I_{i,j-k} - g(k)$ ( Cas 2, figure 2.9 ).

$$\begin{aligned} \Delta_{i,j} \quad &\geq \quad D_{i,j} + 2.ge + S_{i,j} \\ D_{i,j} - \Delta_{i,j} \quad &\leq \quad -(2.ge + m) \end{aligned}$$

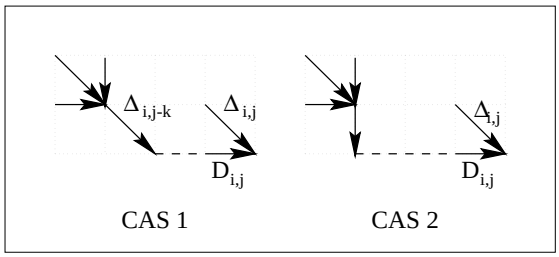



FIG. 2.9 - **Les deux cas possibles** *lorsque* $D_{i,j} \geq \Delta_{i,j}$

Dans le cas où $D_{i,j} \geq \Delta_{i,j}$, on peut résumer en écrivant :

$$D_{i,j} - \Delta_{i,j} \quad \leq \quad M - m \tag{2.18}$$

En prenant en compte les relations 2.17 et 2.18, on peut écrire que :

$$\mid D_{i,j} - \Delta_{i,j} \mid \quad \leq \quad M + \max(2.g(1), -m) \tag{2.19}$$

De même, on a :

$$\mid I_{i,j} - \Delta_{i,j} \mid \quad \leq \quad M + \max(2.g(1), -m) \tag{2.20}$$

Puisqu'on sait que l'écart entre deux valeurs que l'on compare dans l'algorithme de Smith & Waterman, est inférieur à $\{M + \max(2.g(1), -m)\}$ (Cf. éq. 2.19 et 2.20), on en déduit que tant qu'aucune valeur ne dépasse

$$32767 - M - max \begin{pmatrix} 2.g(1), \\ -m \end{pmatrix}$$

l'algorithme donne un résultat valide.

## 2.3 Cas de l'algorithme de recherche de motifs avec erreurs

L'algorithme de Pattern Matching avec erreurs est aussi fondé sur la programmation dynamique. Des algorithmes plus rapides ont été développés (Cf. annexe B) : certains reposent sur la programmation dynamique et utilisent des masques de bits pour accélérer le calcul, d'autres ne se fondent pas sur le même type d'algorithmes et ne donnent pas le même résultat.

Le problème est la recherche des occurrences inexactes d'un mot $x$ dans un texte $t$. Par occurrences inexactes, nous entendons que le motif apparaît soit de manière exacte, soit avec un petit nombres d'erreurs qui sont dans ce cas-là, les substitutions, les insertions ou les délétions. Lorsqu'on utilise les algorithmes de Baeza-Yates & Gonnet ou de Manber & Wu (Cf. annexe B, sections B.2.1 et B.2.3), on doit fournir un nombre maximum d'erreurs permises, et l'algorithme renvoie la position de toutes les occurrences inexactes du mot recherché. Pour chaque occurrence trouvée, on ne connaît pas le nombre exact d'erreurs. Cela peut être très gênant lorsqu'on cherche à classer les différentes occurrences.

Nous nous intéressons ici à l'algorithme initial de recherche de mot avec erreurs, c'est-à-dire celui qui fait appel à la programmation dynamique. Une étude similaire à celle du paragraphe précédent permet de simplifier cet algorithme de Pattern Matching. Il s'agit clairement d'une variante de l'algorithme de Needleman & Wunsch, où

– la matrice de substitution a uniquement des 0 sur la diagonale, et des 1 ailleurs,

- la fonction de pénalisation des gaps est linéaire ($gapo = gape = 1$),
- il ne s'agit plus de prendre le maximum entre la diagonale, la verticale et l'horizontale, mais d'en prendre le minimum.

Les conditions d'initialisation sont légèrement différentes, pour prendre en compte qu'une délétion au début du texte n'est pas pénalisante, mais que celle au début du mot recherché l'est. L'algorithme n'est plus symétrique dans la mesure où les deux séquences ne jouent plus le même rôle. Soit $B[1,m]$ le motif recherché et $A[1,n]$ la séquence dans laquelle on recherche le motif $B$. L'algorithme consiste à calculer toutes les valeurs de la matrice *des scores* $M_{i,j}$ dont la relation de récurrence est donnée par :

$$M_{i,j} = min \begin{pmatrix} M_{i-1,j-1} + \delta_{i,j}, \\ M_{i-1,j} + 1, \\ M_{i,j-1} + 1 \end{pmatrix} \qquad (2.21)$$

$$avec \;\; \delta_{i,j} = \begin{cases} 1 & si \quad A[i] \neq B[j] \\ 0 & si \quad A[i] = B[j] \end{cases}$$

Les initialisations sont données par :

$$\begin{array}{lcll} M_{0,0} & = & 0 & \\ M_{0,j} & = & 0 & j \in [1,m] \\ M_{i,0} & = & i & i \in [1,n] \end{array}$$

Le nombre calculé $M_{n,j}$ représentera le nombre minimum d'erreurs nécessaires pour passer du mot recherché $x$ à un sous-mot terminant à la position $j$ du texte $t$.

### 2.3.1 Relation entre les valeurs $M_{i,j}$ et $M_{i+1,j-1}$

- Première inégalité :

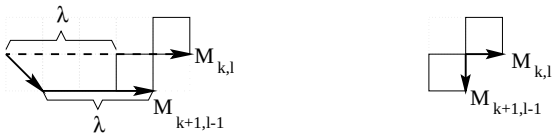



Figure A Figure B

FIG. 2.10 - **Algorithme de pattern matching avec erreurs :** *majoration de $M_{k,l}$ en fonction de $M_{k+1,l-1}$ ($\lambda$ peut être nul).*

- Dans le premier cas ( figure 2.10-A ), il existe $0 \leq \lambda \leq l-2$, tel que le chemin ayant permis d'obtenir $M_{k+1,l-1}$ provient de $M_{k,l-1-\lambda}$. Soit $\delta$ la valeur retenue pour la diagonale : $\delta = 0$ si les deux lettres sont identiques, 1 sinon. On peut écrire :

$$\begin{array}{rcl} M_{k,l} & \leq & M_{k+1,l-1} - \lambda - \delta + \lambda + 2 \\ M_{k,l} - M_{k+1,l-1} & \leq & 2 - \delta \leq 2 \end{array}$$

– Dans le deuxième cas ( figure 2.10-B ), $M_{k+1,l-1}$ provient de $M_{k,l-1}$ et on a :

$$\begin{aligned} M_{k+1,l-1} &= M_{k,l-1} + 1 \\ M_{k,l} &\leq M_{k,l-1} + 1 \\ &\leq M_{k+1,l-1} \\ M_{k,l} - M_{k+1,l-1} &\leq 0 \end{aligned}$$

– Dans les deux cas, on a :

$$M_{k,l} - M_{k+1,l-1} \leq 2 \tag{2.22}$$

- Deuxième inégalité :

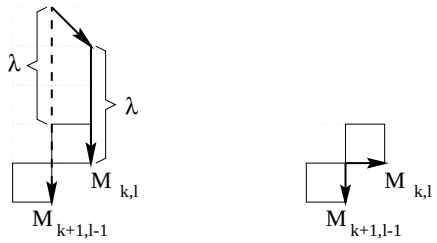



Figure A Figure B

FIG. 2.11 - **Algorithme de pattern matching avec erreurs :** *majoration de $M_{k+1,l-1}$ en fonction de $M_{k,l}$ ($\lambda$ peut être nul).*

– Dans le premier cas ( figure 2.11-A ), il existe $0 \leq \lambda \leq k-1$, tel que le chemin ayant permis d'obtenir $M_{k,l}$ provient de $M_{k-\lambda-1,l-1}$. Soit $\delta$ la valeur retenue pour la diagonale : $\delta = 0$ si les deux lettres sont identiques, 1 sinon. On peut écrire :

$$\begin{aligned} M_{k+1,l-1} &\leq M_{k,l} - \lambda - \delta + \lambda + 2 \\ M_{k+1,l-1} - M_{k,l} &\leq 2 \end{aligned}$$

– Dans le deuxième cas ( figure 2.11-B ), $M_{k,l}$ provient de $M_{k,l-1}$ et on a :

$$\begin{aligned} M_{k,l} &= M_{k,l-1} + 1 \\ M_{k+1,l-1} &\leq M_{k,l-1} + 1 \\ &\leq M_{k,l} \\ M_{k+1,l-1} - M_{k,l} &\leq 0 \end{aligned}$$

– Dans les deux cas, on a :

$$M_{k+1,l-1} - M_{k,l} \leq 2 \tag{2.23}$$

- En mettant en commun les équations 2.22 et 2.23, on a :

$$|M_{i,j} - M_{i+1,j-1}| \leq 2 \tag{2.24}$$

### 2.3.2 Relations entre les valeurs voisines pour l'algorithme de pattern matching avec erreurs

**Propriété 2.3.1** *Pour tout couple d'indices $(k,l)$, on a :*

$$\begin{cases} |M_{k,l} - M_{k-1,l-1}| & \leq & 1 & (Diagonale) \\ |M_{k,l} - M_{k,l-1}| & \leq & 1 & (Horizontale) \\ |M_{k,l} - M_{k-1,l}| & \leq & 1 & (Verticale) \end{cases} \tag{2.25}$$

**Démonstration :**

On va montrer par récurrence la propriété précédente. On pose comme hypothèse de récurrence les trois inégalités de la propriété.

- L'hypothèse est vraie pour $(k,l) = (1,1)$. De plus, on a :
  - $|M_{0,l} - M_{0,l-1}| = 0 \leq 1$ pour $l \geq 1$
  - $|M_{k,0} - M_{k-1,0}| = 1 \leq 1$ pour $k \geq 1$
- Supposons l'hypothèse vraie pour tout couple d'indices $(i,j)$ tel que :

$$\begin{array}{lllll} soit & j & \leq & l-1 & \\ soit & j & = & l & et \quad i \leq k \end{array}$$

  On parcourt la matrice dans le même ordre que l'algorithme.
  Alors :
  - cas de la nouvelle colonne : case $(0, l+1)$. L'inégalité concernant l'horizontale est vérifiée à cause des initialisations. Les deux autres inégalités n'ont pas à être vérifiées.
  - cas général : case $(k+1, l)$. On a la relation de récurrence suivante :

$$M_{k+1,l} = \min\Big(M_{k,l-1} + \delta, M_{k,l} + 1, M_{k+1,l-1} + 1\Big)$$

    On doit vérifier les trois inégalités (Cf. éq. 2.25).
    1. $M_{k+1,l} - M_{k,l} = \min\Big(\underbrace{M_{k,l-1} - M_{k,l} + \delta}_{a}, 1, \underbrace{M_{k+1,l-1} - M_{k,l} + 1}_{b}\Big)$

       Le terme $a$ est compris entre $-1$ et 2, et on a $-1 \leq b \leq 3$ d'après l'hypothèse de récurrence et d'après l'équation 2.24. Donc

$$-1 \leq M_{k+1,l} - M_{k,l} \leq 1.$$

    2. $M_{k+1,l} - M_{k+1,l-1} = \min\Big(\underbrace{M_{k,l-1} - M_{k+1,l-1} + \delta}_{a}, \underbrace{M_{k,l} - M_{k+1,l-1} + 1}_{b}, 1\Big)$

       On a : $-1 \leq a \leq 2$ et $-1 \leq b \leq 3$ d'après l'hypothèse de récurrence et d'après l'équation 2.24. Donc

$$-1 \leq M_{k+1,l} - M_{k+1,l-1} \leq 1.$$

3. $M_{k+1,l} - M_{k,l-1} = \min\Big(\underbrace{\delta}_{a}, \underbrace{M_{k,l} - M_{k,l-1} + 1}_{b}, \underbrace{M_{k+1,l-1} - M_{k,l-1} + 1}_{c}\Big)$

   On a : $0 \leq a \leq 1$, $0 \leq b \leq 2$ et $0 \leq c \leq 2$. Donc,

$$-1 \leq M_{k+1,l} - M_{k,l-1} \leq 1.$$

- L'hypothèse est donc vérifiée pour tout (k,l).

□

### 2.3.3 Implications

L'algorithme de la recherche d'un motif avec $k$-erreurs est un algorithme de programmation dynamique en $O(m \times n)$ (Cf. figure 2.12-A).

**Algorithme Classique ( A )**

```
int kerr(unsigned char *text, int n,
         unsigned char *pattern, int m)

    int i,j, tj,tj1;
    int nb_err = m;

    T[0] = 0;
    for(j=1; j<=m; j++) T[j] = j;

    for(i=1; i<=n; i++)
        tj1 = 0;
        for(j=1; j<=m; j++)
            tj = T[j];

            if (pattern[m-j] != text[n-i])  test 1
               tj1++;
            if (tj1 > tj+1)                 test 2
               tj1 = tj+1;
            if (tj1 > T[j-1]+1)             test 3
               tj1 = T[j-1]+1;

            T[j] = tj1;
            tj1 = tj;

        if (T[m] < nb_err)  nb_err = T[m];

    return(nb_err);
```

**Algorithme modifié ( B )**

```
int kerr(unsigned char *text, int n,
         unsigned char *pattern, int m)

    int i,j, tj,tj1;
    int nb_err = m;

    T[0] = 0;
    for(j=1; j<=m; j++) T[j] = j;

    for(i=1; i<=n; i++)
        tj1 = 0;
        for(j=1; j<=m; j++)
            tj = T[j];

            if (pattern[m-j] != text[n-i])
                tj1++;
                if (tj1 > tj+1)
                   tj1 = tj+1;
                else if (tj1 > T[j-1]+1)
                   tj1 = T[j-1]+1;

            T[j] = tj1;
            tj1 = tj;

        if (T[m] < nb_err)  nb_err = T[m];

    return(nb_err);
```

FIG. 2.12 - **Algorithme de pattern matching avec erreurs**

Le cœur de l'algorithme est constitué des trois tests nécessaires pour le calcul de l'équation 2.21. Cependant, le paragraphe précédent montre qu'on peut diminuer le nombre moyen de comparaisons. Lorsqu'on a l'égalité des lettres, les deux autres tests sont superflus. Supposons qu'on ait l'égalité des lettres $A[k]$ et $B[l]$. On a $M_{k,l} = M_{k-1,l-1}$. Puisque $|M_{k-1,l-1} - M_{k-1,l}| \leq 1$, on a

$$\begin{array}{ccccc} -1 & \leq & M_{k,l} - M_{k-1,l} & \leq & 1 \\ M_{k-1,l} - 1 & \leq & M_{k,l} & \leq & 1 + M_{k-1,l} \end{array}$$

et donc on ne peut avoir $M_{k,l} > 1 + M_{k-1,l}$. Le test numéro 2 (Cf. figure 2.12) n'est donc jamais vérifié. De la même manière, on montre que le troisième test est inutile.

Supposons maintenant que les deux lettres concernées sont différentes. Les deux autres tests sont exclusifs. En effet,

$$M_{k,l} = \min\Big(M_{k-1,l-1}, M_{k,l-1}, M_{k-1,l}\Big) + 1$$

Si l'horizontale a un meilleur score que la diagonale, c'est-à-dire si $\min(M_{k-1,l-1}, M_{k,l-1}) = M_{k,l-1}$, peut-on avoir une verticale dont le score soit strictement inférieur à celui de l'horizontale ? Peut-on avoir $M_{k-1,l} < M_{k,l-1}$ ?

Si c'était le cas, on aurait $M_{k-1,l} < M_{k,l-1} < M_{k-1,l-1}$, c'est-à-dire que :

$$\begin{aligned} M_{k-1,l} &= M_{k,l-1} - 1 \\ M_{k,l-1} &= M_{k-1,l-1} - 1 \\ M_{k-1,l} &= M_{k-1,l-1} - 2 \end{aligned}$$

Ce qui est impossible à cause de la propriété 2.3.1.

En mettant en œuvre l'algorithme présenté en figure 2.12 (Algorithme B), nous arrivons à améliorer de façon significative les performances de l'algorithme de pattern matching avec erreurs (Cf. tableau 2.1).

| | Algorithme Classique | Algorithme modifié |
|---|---|---|
| Temps | 6 mn 04,91 | 3 mn 38,69 s |
| MMC/s [a] | 7,493 | 12,502 |

TAB. 2.1 - **Performances.** *On fait la comparaison banque contre banque d'une petite banque de 10 motifs (109 résidus) contre SwissProt Rel. 35 (25 083 768 résidus). L'algorithme implémenté sort de la boucle dès qu'une occurrence exacte du motif est détectée. Les deux algorithmes sont implémentés dans LASSAP ([48]). Les résultats sont ceux obtenus sur serveur SUN 4000, en n'utilisant qu'un seul processeur.*

[a] Million Matrix Cells per second.

## 2.4 Influence des paramètres sur les résultats de l'algorithme

L'algorithme de Smith & Waterman fait intervenir un grand nombre de paramètres qui sont les pénalisations des insertions/délétions et les coûts de substitution d'une lettre par une autre. Dans le cas de la recherche des similarités locales entre protéines, il y a 212 paramètres indépendants : deux pour les pénalisations des insertions/délétions et 210 dans la matrice de substitution (puisque la matrice est de taille 20 et qu'elle est symétrique).

Le choix de ces paramètres influe beaucoup sur les résultats obtenus, mais les effets de ce choix ne sont pas bien compris. Certes certains éléments ont bien été décrits, comme, par exemple, le changement de phase entre un comportement linéaire et un comportement logarithmique [127].

D'autres études portent sur la décomposition de l'espace des paramètres [58] : puisque le score d'alignement optimal et même l'alignement sont des éléments discrets, un score optimal pour des paramètres fixés est optimal pour toute une région dans l'espace des paramètres. On cherche donc à découper l'espace des paramètres en différentes zones, telles que dans chacune d'elles l'alignement maximal est identique. Dans chacune de ces régions, le score est une fonction linéaire des paramètres.

Ainsi, les régions sont bornées par des intersections d'hyperplans. Une telle décomposition de l'espace des paramètres n'est pas accessible pour l'instant dans le cas général. Les études faites se bornent à des cas très simples, où la matrice est dégénérée, c'est-à-dire où ses éléments sont 1 ou $-\delta$ suivant qu'il s'agit d'un élément diagonal ou non.

Pourtant ces décompositions pourraient permettre de définir un indice de stabilité pour l'alignement. Plus l'alignement se trouve près des hyperplans qui définissent ce même alignement, plus l'alignement est instable. Deux indices pourraient donner une estimation de la stabilité :

- la distance à la plus proche frontière,
- le «*volume*» de la zone correspondant à l'alignement effectivement trouvé.

Un tel indice apporterait une information supplémentaire pour choisir le bon alignement parmi les $n$ alignements sous-optimaux. Un alignement instable, c'est-à-dire un alignement qu'on ne retrouve pas lorsqu'on perturbe les paramètres, est moins informatif qu'un alignement sous-optimal stable, puisque ce dernier est préservé lorsque la matrice de substitution et les pénalités des insertions/délétions changent.

**Relations entre la longueur de l'alignement et le couple (go,ge)** L'analyse de la dépendance de l'alignement en fonction de l'ensemble des paramètres semble hors de portée aujourd'hui. Restreignons-nous à la dépendance de la longueur de l'alignement en fonction des pénalités d'insertion/délétion, à matrice fixe.

### Le cas Needleman & Wunsch

Dans le cas de l'alignement global, on a rapidement un encadrement de la longueur de l'alignement. En effet, dans le cas où la plus petite des deux séquences est entièrement comprise dans la plus grande, l'alignement de comprend pas d'insertions/délétions au milieu de l'alignement, et les insertions/délétions se trouvent uniquement aux deux bords de l'alignement. La longueur de l'alignement est alors $\max\{m, n\}$. A l'autre extrême, si les deux séquences sont totalement étrangères l'une à l'autre, l'alignement peut ne pas contenir d'appariement. Dans ce cas la longueur de l'alignement est $m + n$.

| | |
|---|---|
| longueur minimale | $\max\{m, n\}$ |
| longueur maximale | $m + n$ |

**Propriété 2.4.1** *Si $go = ge$, le score est une fonction décroissante de $ge$.*

> **Preuve** : Pour un alignement constant, le score est une fonction décroissante de $ge$.
> De plus, lorsqu'il y a un changement d'alignement, le score reste une fonction continue.
> En effet, pour chacun des alignements le score est une fonction linéaire donc continue,
> et la fonction max est aussi continue.
> Les deux arguments précédents donnent le résultat. □

**Propriété 2.4.2** *La longueur est une fonction décroissante de ge ($go \geq ge$).*

**Preuve** : Supposons qu'on ait un alignement $A_1$ de score $s_1$ avec $n_{v1}$ insertions, $n_{h1}$ délétions pour une pénalisation des insertions/délétions donnée par $go_1$ et $ge_1$. On obtient avec $ge_2 < ge_1$ un autre alignement $A_2$ de score $s_2$ avec $n_{v2}$ insertions et $n_{h2}$ délétions. la longueur $l$ de l'alignement est :

$$l = n_h + n_v + n_d$$

où $n_h$, $n_v$, $n_d$ sont les nombres de chemins horizontaux, verticaux et diagonaux. On a la relation :

$$2 \times n_d + n_h + n_v = n + m$$

En fonction du nombre d'insertions/délétions, trois cas sont possibles. Pour deux de ces cas, la longueur du nouvel alignement est supérieure ou égale à la longueur de l'alignement initial. Le troisième cas est impossible.

- Si le nombre d'insertions/délétions est resté le même, le nombre d'appariements n'a pas changé non plus (il s'agit de Needleman & Wunsch). La longueur est la même.
- Supposons que le nombre d'insertions/délétions a augmenté. Puisqu'il s'agit de l'alignement global, on a :

$$\begin{aligned} 2 \times l &= \underbrace{n_d + n_h}_{n} + \underbrace{n_d + n_v}_{m} + \underbrace{n_h + n_v}_{n_i} \\ 2 \times l &= n + m + n_i \end{aligned}$$

  où $m$ et $n$ sont les longueurs respectives des deux séquences, $n_i = n_{hi} + n_{vi}$ le nombre d'insertions/délétions pour l'alignement $i$, et $l$ la longueur de l'alignement. Si $n_i$ augmente, la longueur $l$ augmente aussi.
- Supposons $n_2 < n_1$.
  L'alignement $A_1$ avec $ge_1$ a un score $s(A_1, ge_1) = D_1 - (n_1 - 1).ge_1 - n_{g1}.go$.
  L'alignement $A_1$ avec $ge_2$ a un score $s(A_1, ge_2) = D_1 - (n_1 - 1).ge_2 - n_{g1}.go$.
  L'alignement $A_2$ avec $ge_1$ a un score $s(A_2, ge_1) = D_2 - (n_2 - 1).ge_1 - n_{g2}.go$.
  L'alignement $A_2$ avec $ge_2$ a un score $s(A_2, ge_2) = D_2 - (n_2 - 1).ge_2 - n_{g2}.go$.
  où $D_i$ est le poids des appariements le long du chemin correspondant à l'alignement $A_i$.
  On a les relations suivantes :

$$s(A_1, ge_2) \leq s(A_2, ge_2) \quad (2.26)$$
$$s(A_1, ge_1) \leq s(A_1, ge_2) \quad (2.27)$$
$$s(A_2, ge_1) \leq s(A_2, ge_2) \quad (2.28)$$

  L'équation 2.26 est vraie parce que l'alignement optimal est $A_2$ dans le cas de $ge_2$. Les équations 2.27 et 2.28 le sont parce qu'on diminue la pénalité des insertions/délétions.
  Montrons que si $n_2 < n_1$ on a : $s(A_1, ge_1) \leq s(A_2, ge_1)$, ce qui sera incompatible avec le fait que $A_1$ est l'alignement optimal lorsque la pénalisation des insertions/délétions est $ge_1$.

$$
\begin{aligned}
& s(A_1, ge_1) - s(A_2, ge_1) = \\
& \quad = (D_1 - D_2) + (n_2 - n_1).ge_1 + (n_{g2} - n_{g1}).go \\
& \quad = \underbrace{(D_1 - D_2) + (n_2 - n_1).ge_2 + (n_{g2} - n_{g1}).go}_{\leq 0 \quad (Cf.\ éq.\ 2.26)} + \underbrace{(n_2 - n_1)}_{<0} . \underbrace{(ge_1 - ge_2)}_{>0} \\
& \quad \leq 0
\end{aligned}
$$

□

Dans le cas de l'alignement local on a affaire à une autre difficulté liée au fait que ni le début de l'alignement ni la fin ne sont fixés. Même si dans la majorité des cas, la longueur est une fonction décroissante des pénalités des insertions/délétions, on peut trouver des contre-exemples.

## 2.5 Relation entre les algorithmes SW et NW

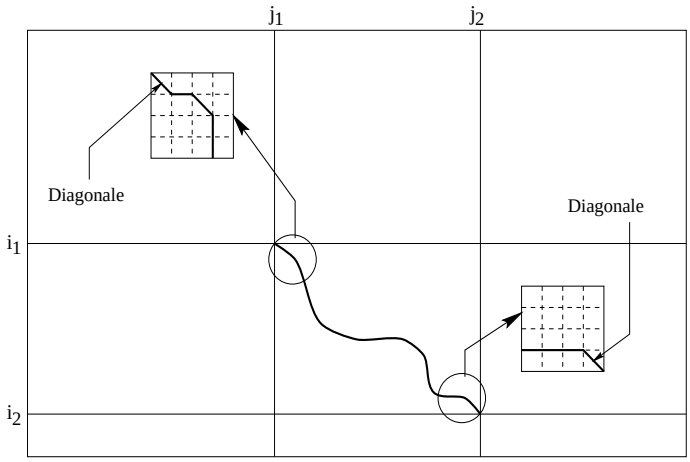



FIG. 2.13 - **L'algorithme de Smith & Waterman** *recherche les sous-séquences $A[i_1+1, i_2]$ et $B[j_1+1, j_2]$ pour lesquelles le score Needleman & Wunsch est le plus élevé. Il est à noter que les bords de l'alignement sont des diagonales de coût positif (Cf. remarque p. 27), puisque si cette condition n'était pas vérifiée, il existerait un alignement de meilleur score : ce même alignement, auquel on aurait supprimé les extrémités de coûts négatifs (insertions/délétions et substitutions négatives).*

On cherche les sous-séquences de support $I = [i_1, i_2]$ et $J = [j_1, j_2]$, c'est-à-dire les sous-séquences $A[i_1, i_2]$ et $B[j_1, j_2]$ pour lesquelles le score NW est maximum. L'algorithme de Smith &

Waterman s'exprime par la récurrence suivante :

$$\begin{cases} H(i,j) &= \max\Big(S(i,j-1)-go\ ,\ H(i,j-1)-ge\Big) \\ V(i,j) &= \max\Big(S(i-1,j)-go\ ,\ V(i-1,j)-ge\Big) \\ D(i,j) &= S(i-1,j-i)+\sigma(i,j) \\ S(i,j) &= \max\Big(V(i,j)\ ,\ H(i,j)\ ,\ S(i,j)\ ,\ 0\Big) \end{cases}$$

Montrons que l'algorithme de Smith & Waterman résout bien le problème posé. Posons $N_{k,l}(i,j)$ le score NW en $(i,j)$ pour les sous-séquences $A[k,N_A]$ et $B[l,N_B]$, où $N_A$ et $N_B$ sont les longueurs respectives des séquences $A$ et $B$.

**Propriété 2.5.1** *On a :*

$$S(i,j) \geq N_{0,0}(i,j) \quad \textit{pour tout} \quad i \geq k \quad \textit{et} \quad j \geq l$$

**Démonstration** : La preuve est immédiate, puisque l'algorithme SW prend le maximum avec 0, ce qui ne peut qu'augmenter le score. □

**Propriété 2.5.2** *Soit $(k,l)$ tel que $S(k,l)=0$. Alors*

$$S(i,j) \geq N_{k,l}(i,j) \quad \textit{pour tout} \quad i \geq k \quad \textit{et} \quad j \geq l$$

**Démonstration** : Le démonstration se fait par récurrence sur le numéro de la diagonale. Dans la matrice des scores, on considère successivement les diagonales $i+j=d, \forall d \in [k+l, k+l+1, \ldots, n+m]$, où $i \geq k$ et $j \geq l$. $d$ est appelé le numéro de la diagonale.

- La propriété est vraie pour tout couple $(k,j)$ tel que $j \geq l$ et pour tout couple $(i,l)$, tel que $i \geq k$. En particulier, la propriété est vraie pour la diagonale $k+l$.
- Supposons l'hypothèse de récurrence vraie pour toutes les diagonales de numéro strictement inférieur à $d$. Montrons qu'elle est vraie sur la diagonale $d$. Soit $(i,j)$ un couple d'indice appartenant à la diagonale $d$. Alors :

$$\begin{aligned} S(i,j) &= \max\Big(S(i-1,j-1)+\sigma(i,j)\ ,\ V(i,j)\ ,\ H(i,j)\ ,\ 0\Big) \\ N_{k,l}(i,j) &= \max\Big(N_{k,l}(i-1,j-1)+\sigma(i,j)\ ,\ N^V_{k,l}(i,j)\ ,\ N^H_{k,l}(i,j)\Big) \end{aligned}$$

  On a par hypothèse de récurrence les trois inégalités suivantes :

  $S(i-1,j-1) \geq N_{k,l}(i-1,j-1)$
  $V(i,j) \geq N^V_{k,l}(i,j)$ et
  $H(i,j) \geq N^H_{k,l}(i,j)$.

  On en conclut que $S(i,j) \geq N_{k,l}(i,j)$. Et la propriété est vraie pour toute la diagonale $d$.
- La propriété est donc vraie pour toutes les diagonales $d = k+l, k+l+1, \ldots, n+m$.□

**Propriété 2.5.3** *Soit $(m,n)$ un double indice de la matrice des scores Smith & Waterman tel que le score SW soit strictement positif : $S(m,n) > 0$. Si on trouve comme début d'alignement local la substitution entre les lettres $A[k]$ et $B[l]$, avec $k < m$ et $l < n$, alors $S(m,n)$ correspond au score NW des sous-séquences $A[k,m]$ et $B[l,n]$.*

**Démonstration** : Montrons par récurrence que sur ce chemin, les valeurs $S(i,j) = N_{k,l}(i,j)$.

- La propriété est vraie pour la position 1 de l'alignement correspondant à $(k,l)$ : $S(k,l) = N_{k,l}(k,l) = \sigma(k,l) \geq 0$.
- Supposons que la propriété est vraie jusqu'à la position $t$ de l'alignement correspondant au double indice $(i',j')$.
  - Supposons que l'alignement SW passe par une substitution pour la position $t+1$ de l'alignement correspondant à $(i'',j'')$. $i'' = i'+1$ et $j'' = j'+1$.

    $$S(i'',j'') = \max(A', B', C'+\gamma, 0) = C'+\gamma$$

    où $\gamma = S(A[i''], B[j''])$.
    Regardons le score $N_{k,l}(i'',j'')$.

    $$N_{k,l}(i'',j'') = \max(A, B, C+\gamma)$$

    Par la propriété 2.5.2, on a $A' \geq A$, $B' \geq B$ et $C' \geq C$. On a donc

    $$C+\gamma \leq \max(A,B,C+\gamma) \leq \max(A',B',C'+\gamma) = C'+\gamma$$

    Or par hypothèse de récurrence, on a $C = C'$. On en déduit $N_{k,l}(i'',j'') = C+\gamma$. L'alignement NW en $(i'',j'')$ se termine par une diagonale.
  - Supposons maintenant que l'alignement SW passe par une verticale pour la position $t+1$ de l'alignement correspondant à $(i'',j'')$ :

    $$S(i'',j'') = \max(A', B', C'+\gamma, 0) = A' = S(i''',j''') - go - s.ge$$

    Le score Needleman & Wunsch s'écrit $N_{k,l}(i'',j'') = \max(A,B,C+\gamma)$. Par la propriété 2.5.2, on a $A' \geq A$, $B' \geq B$ et $C' \geq C$. On a donc :

    $$A \leq \max(A,B,C+\gamma) \leq \max(A',B',C'+\gamma) = A'$$

    Or par hypothèse de récurrence, on a $S(i''',j''') = N_{k,l}(i''',j''')$. On en déduit que $A = A'$ puis que $N_{k,l}(i'',j'') = A$. L'alignement NW en $(i'',j'')$ se termine par une verticale.
  - Le dernier cas (horizontale) se traite de la même manière.

  La propriété est donc vraie pour la position $t+1$ de l'alignement.
- La propriété est vraie jusqu'à la position $(m,n)$.

□

On a donc $S(m,n) = N_{k,l}(m,n)$.
On en déduit qu'aucun chemin de $(k,l)$ à $(m,n)$ n'a un score supérieur à $S(m,n)$.
Le score obtenu par SW est égal à celui obtenu par NW sur la restriction à $[k,m] \times [l,n]$.

**Propriété 2.5.4** *Si $(i_1,j_1)$ est le début de l'alignement optimal trouvé par l'algorithme SW, tous les chemins partant de n'importe où et arrivant en $(i_1,j_1)$ ont un score NW négatif ou nul.*

**Démonstration** : Par l'absurde. Si ce n'était pas le cas, il existerait $(k,l)$ tel que $N_{k,l}(i_1,j_1) > 0$. Le long de cet alignement, on aurait $S(i,j) \geq N_{k,l}(i,j)$, et donc $S(i_1,j_1) > 0$. □

**Propriété 2.5.5** *Soit $(i_1,j_1)$ le début de l'alignement SW. Soit $(i_2,j_2)$ la fin de l'alignement SW. Le chemin SW trouvé est le meilleur parmi tous les alignements NW partant de $(i'_1 \leq i_1, j'_1 \leq j_1)$ passant par $(i_1,j_1)$ et se terminant en $(i'_2 \geq i_1, j'_2 \geq j_1)$.*

**Démonstration** : Considérons un alignement passant par $(i_1,j_1)$. La partie de l'alignement à gauche de $(i_1,j_1)$ a un score NW $\leq 0$. La partie à droite de $(i_1,j_1)$ a un score NW inférieur ou égal à $S(i_2,j_2)$, car $S(i_2,j_2)$ est le maximum de la matrice des scores.

Donc, le chemin trouvé est le meilleur parmi tous ceux passant par $(i_1,j_1)$. □

**Propriété 2.5.6** *Tous les autres chemins (ne passant pas par $(i_1,j_1)$) ont un score NW inférieur.*

**Démonstration** : Considérons un chemin ne passant pas par $(i_1,j_1)$. Il passe en $(k,l)$. Le meilleur chemin passant par $(k,l)$ se termine en $(i'_2,j'_2)$ et a un score de $S(i'_2,j'_2) \leq S(i_2,j_2)$. □

On peut aussi dire simplement que $S(i,j)$ est le score NW du meilleur chemin finissant en $(i,j)$.

- On remonte dans le tableau jusqu'à la position $(i_1,j_1)$ qui est le début de l'alignement optimal se terminant en $(i,j)$. On n'est pas passé par 0. Donc $SW(i,j) = NW_{i_1,j_1}(i,j)$.
- Peut-il y avoir un autre point de départ $(i'_1,j'_1)$ pour lequel le score du chemin de $(i'_1,j'_1)$ à $(i,j)$ soit strictement supérieur à $S(i,j)$ (Cf. figure 2.14)? Supposons qu'un tel alignement existe. Comme les deux chemins partagent au moins la diagonale de la fin de l'alignement, il existe une position $(k,l)$ telle que

$$N_{i'_1,j'_1}(k,l) > N_{i_1,j_1}(k,l)$$

En $(k,l)$, on a $S(k,l) = N_{i'_1,j'_1}(k,l) > N_{i_1,j_1}(k,l)$. Ce qui est impossible car pour l'alignement commençant en $(i,j)$ on a, d'après la propriété 2.5.3 $S(k,l) = N_{i_1,j_1}(k,l)$.

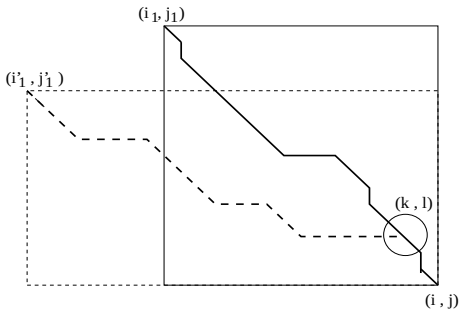



FIG. 2.14 - **Le score** *$S(i,j)$ est le score Needleman & Wunsch du meilleur chemin finissant en $(i,j)$*

A la vue de ces différentes propriétés, on en déduit que le score SW répond bien à la question initiale, à savoir la recherche des deux sous-séquences qui maximisent le score NW.

## 2.6 Dénombrement des alignements

Etant données deux séquences $A$ et $B$ de taille respective $n$ et $m$, combien d'alignements sont possibles ? Deux cas sont à distinguer, les alignements globaux et les alignements locaux.

### 2.6.1 Nombre d'alignements globaux pour des séquences de longueurs fixes

**Première approche :** Un alignement est une mise en correspondance de plusieurs lettres des séquences A et B. On peut avoir pour les séquences $A = atagc$ et $B = aagcc$, les alignements suivants : $\begin{bmatrix} a\ t\ a\ g\ c \\ a\ a\ g\ c\ c \end{bmatrix}$ ou bien $\begin{bmatrix} a\ t\ a\ g\ c\ - \\ a\ -\ a\ g\ c\ c \end{bmatrix}$

Pour compter le nombre d'alignements possibles, il suffit de connaître le nombre d'alignements comprenant exactement une substitution, 2 substitutions, jusqu'au nombre d'alignements contenant k substitutions. $k \leq min(m,n)$. Le nombre total d'alignements globaux vaut alors :

$$N_{global} = \sum_{k=0}^{min(m,n)} C_m^k C_n^k = C_{m+n}^{min(m,n)} \tag{2.29}$$

On peut considérer que les alignements du type de $\begin{bmatrix} a\ t\ -\ a\ g\ c \\ a\ -\ a\ g\ c\ c \end{bmatrix}$ ne sont pas réellement des alignements spécifiques puisqu'aucune information ne nous permet de choisir entre les deux possibilités suivantes : $\begin{bmatrix} a\ t\ -\ a\ g\ c \\ a\ -\ a\ g\ c\ c \end{bmatrix}$, $\begin{bmatrix} a\ -\ t\ a\ g\ c \\ a\ a\ -\ g\ c\ c \end{bmatrix}$. Une fois connus les emplacements des substitutions, l'existence et la position des insertions/délétions ne sont pas déterminées exactement. Ainsi, on peut considérer que les deux alignements précédents sont deux représentations possibles d'un seul alignement. Dans ce cas, la formule (2.29) donnée par Waterman et Vingron [133] donne une surévaluation du nombre d'alignements.

On peut cependant considérer que les deux alignements précédents sont différents. L'alignement $\begin{bmatrix} a\ t\ -\ a\ g\ c \\ a\ -\ a\ a\ g\ c \end{bmatrix}$ peut signifier que les deux séquences qui entrent en jeu sont issues d'une même séquence ancêtre qui serait : $[ataagc]$. L'expression précédente (2.29) est alors en dessous du nombre souhaité puisque les alignements : $\begin{bmatrix} a\ t\ -\ a\ g\ c \\ a\ -\ a\ a\ g\ c \end{bmatrix}$ et $\begin{bmatrix} a\ -\ t\ a\ g\ c \\ a\ a\ -\ a\ g\ c \end{bmatrix}$ sont bien distincts. Dans le premier cas, la séquence mère serait $[ataagc]$ alors que dans le second alignements, elle serait $[aatagc]$.

La sous-évaluation du nombre d'alignements par cette méthode est due au fait que l'on ne considère pas les différentes façons d'aligner les séquences entre les substitutions. Les figures 2.15 et 2.16 illustrent cette faiblesse.

**Relation de récurrence :** Stanton et Cowan [120] ont donné une expression exacte du nombre d'alignements entre deux séquences, de longueur $m$ et $n$ respectivement. Soit $G_{m,n}$ le nombre d'alignements possibles entre deux séquences de longueurs $m$ et $n$. Ils introduisent une récurrence :

- pour m=0, il y a un seul alignement, $G_{0,n} = 1,\ \forall n \geq 0$
- pour n=0, il y a un seul alignement, $G_{m,0} = 1,\ \forall m \geq 0$
- pour $m > 0$ et $n > 0$, on a la relation suivante :

$$G_{m,n} = G_{m-1,n-1} + G_{m-1,n} + G_{m,n-1} \tag{2.30}$$

où

- $G_{m-1,n-1}$ représente le nombre d'alignements se terminant par une substitution,

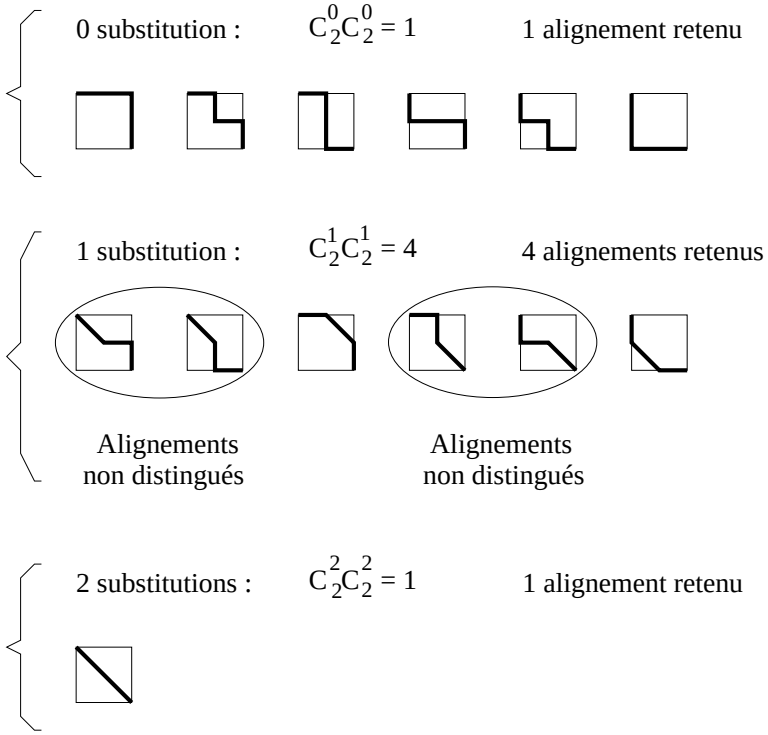



FIG. 2.15 - **Calcul du nombre d'alignements globaux.** *La formule simple citée ci-dessus 2.29, ne distingue pas tous les alignements.*

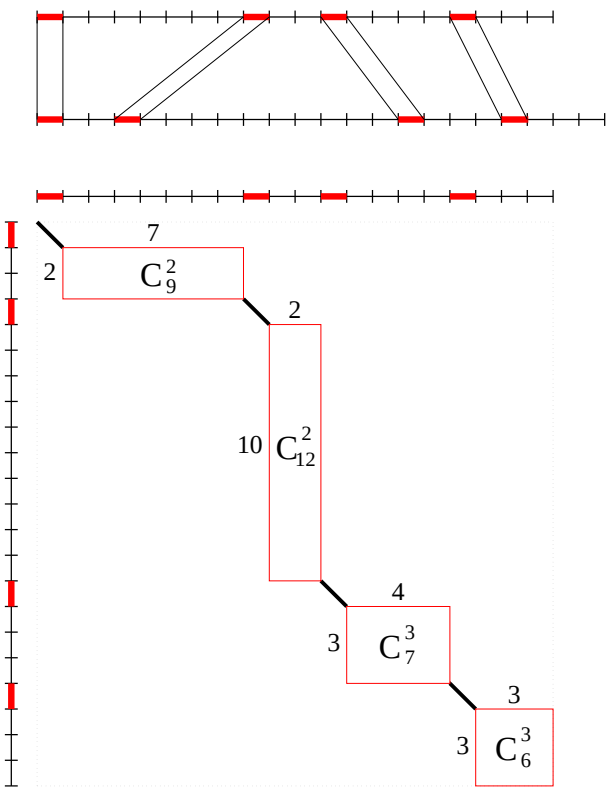



FIG. 2.16 - **Calcul du nombre d'alignements globaux pour des substitutions fixées.** *Soient deux séquences de longueurs respectives* 20 *et* 22. *On considère les alignements comportant les* 4 *substitutions dessinées ci-dessus. On compte les différentes possibilités d'«aligner» les séquences* entre les substitutions. *Entre la* $3^{ème}$ *et la* $4^{ème}$ *substitution, on a* $C_7^3$ *chemins possibles, puisqu'on ne se permet que des horizontales et des verticales. En effet, on a* $3+4=7$ *déplacements élémentaires, parmi lesquels il y a* 3 *déplacements verticaux. Il s'agit de compter le nombre de chemins possibles de longueur* 7, *contenant* 3 *déplacements verticaux :* $C_7^3$.

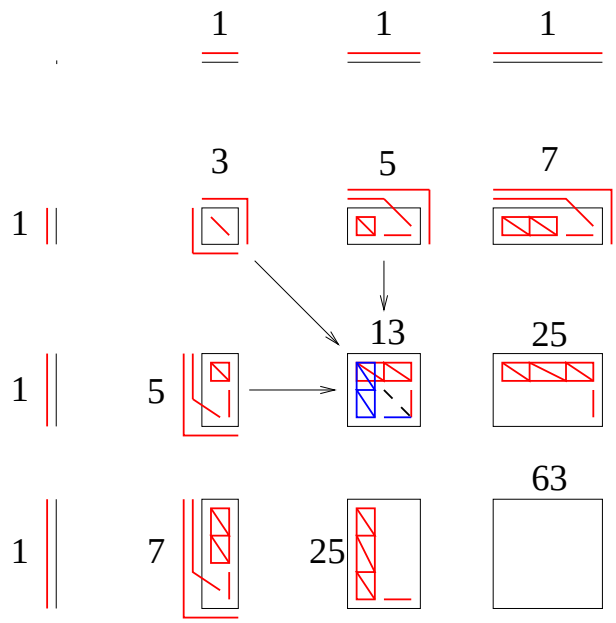



FIG. 2.17 - **Calcul du nombre d'alignements globaux.** *Ce nombre vérifie une relation de récurrence à deux indices.* $G_{m,n} = G_{m-1,n-1} + G_{m-1,n} + G_{m,n-1}$. *Pour deux séquences de longueur 2, il existe 13 alignements possibles.*

- $G_{m-1,n}$ représente le nombre d'alignements se terminant par une insertion de la dernière lettre, et
- $G_{m,n-1}$ représente le nombre d'alignements se terminant par une délétion de la dernière lettre dans la première séquence.

Cette récurrence est illustrée par la figure 2.17.

Les auteurs donnent une expression exacte du nombre d'alignements. Ils montrent par récurrence que :

$$G_{m,n} = \sum_{\alpha=max(n,m)-min(n,m)}^{max(m,n)} C_n^{\alpha}.C_{m+\alpha}^{n}$$

Le résultat final est :

$$G_{m,n} = \sum_{\alpha=0}^{min(m,n)} 2^{\alpha}.C_n^{\alpha}C_m^{\alpha}$$

avec la convention suivante :

$$C_k^l = 0 \; si \; k < l \; ou \; si \; l < 0$$

H.T. Laquer [84] donne une limite asymptotique pour une généralisation de la variable $G_{m,n}$. La généralisation consiste à introduire un paramètre supplémentaire dans la relation de récurrence. L'équation 2.30 devient :

$$G_{m,n} = G_{m-1,n-1} + b.G_{m-1,n} + G_{m,n-1}, \; b \in \mathbb{N}$$

On a alors :

$$G_{m,n} = \sum_{\alpha=0}^{min(m,n)} (1+b)^{\alpha}.C_n^{\alpha}C_m^{\alpha}$$

Il donne une limite asymptotique du comportement de $G_{m,n}$ lorsque $m$ et $n$ tendent simultanément vers $+\infty$.

**Théorème 1** *Posons $x = (1+b)$. Etant donné $\epsilon > 0$ et un intervalle fermé $[\alpha_0;\alpha_1] \subset [0,\infty]$, il existe $N_0$ tel que pour $n > N_0$ et pour $m/n \in [\alpha_0;\alpha_1]$ on a :*

$$\left| \frac{G_{m,n}}{k(m,n)} - 1 \right| < \epsilon$$

*où*

$$\begin{aligned} k(m,n) &= \frac{1}{\sqrt{8\pi}} \left\{ \sqrt{\frac{m}{n}} + \sqrt{\frac{n}{m}} + \left[ \left( \sqrt{\frac{m}{n}} + \sqrt{\frac{n}{m}} \right)^2 - \frac{4(x-1)}{x} \right]^{\frac{1}{2}} \right\} \\ &\quad \times \frac{\psi(\frac{m}{n})^n \psi(\frac{n}{m})^m}{\left( (n+m)^2 - \frac{4(x+1)}{x} nm \right)^{1/4}} \\ \textit{et} & \\ \psi(\alpha) &= 1 - \frac{x}{2} \left[ 1 - \alpha - \left( (1+\alpha)^2 - \frac{4(x-1)}{x}\alpha \right)^{1/2} \right] \end{aligned}$$

H.T. Laquer utilise cette formule pour donner le nombre d'alignements entre 2 séquences, de longueur respective $m$ et $n$. $b = 1$ et $x = 2$. $\psi(\alpha) = \alpha + \sqrt{1+\alpha^2}$ et on a :

$$G_{m,n} \sim \frac{\sqrt{\frac{m}{n}} + \sqrt{\frac{n}{m}} + \sqrt{\frac{m}{n} + \frac{n}{m}}}{\sqrt{8\pi}(n^2+m^2)^{1/4}} \left( \frac{m}{n} + \sqrt{1 + \frac{m^2}{n^2}} \right)^n \left( \frac{n}{m} + \sqrt{1 + \frac{n^2}{m^2}} \right)^m$$

**Expression exacte du nombre d'alignements dans le cas où on ne considère pas les alignements du type $\begin{bmatrix} a & t & - & a & g & c \\ a & - & a & g & c & c \end{bmatrix}$ comportant une insertion et une délétion contiguës.**

Nous avons à évaluer le nombre de chemins possibles pour aller de $(0,0)$ et $(n,m)$ où $m$ et $n$ représentent les longueurs respectives des séquences considérées. Mais, certaines contraintes compliquent le calcul. Il n'y a que 3 déplacements élémentaires :

- un déplacement de vecteur $(1,0)$ (déplacement horizontal),
- un déplacement de vecteur $(0,1)$ (déplacement vertical) et
- un déplacement de vecteur $(1,1)$ (déplacement diagonal).

De plus, un déplacement vertical ne peut avoir lieu après un déplacement horizontal, et vice versa. On ne considère donc que les chemins sans angle droit. Cela revient à dire qu'on ne considère pas les alignements du type $\begin{bmatrix} a & t & - & a & g & c \\ a & - & a & g & c & c \end{bmatrix}$. Dans la pratique, $-go - ge < min_{i,j}(\sigma_{i,j})$. Les algorithmes ne génèrent alors jamais de tels alignements.

Utiliser les trois vecteurs de déplacements élémentaires $(1,0)$, $(0,1)$ ou $(1,1)$ se code par trois lettres : $a$ codera pour $(1,0)$, $b$ pour $(0,1)$ et $c$ codera pour $(1,1)$. Un mot $w$ code un chemin de (0,0) a (n,m) si et seulement si :

$$\begin{cases} |w|_a + |w|_c &= n \\ et & \\ |w|_b + |w|_c &= m \end{cases}$$

Les contraintes «*pas d'angle droit*» s'expriment par : pas de $ab$, pas de $ba$. Autrement dit le problème est de rechercher les mots dans le langage rationnel :

$$L = (a+b+c)^* - (a+b+c)^* ab (a+b+c)^* - (a+b+c)^* ba (a+b+c)^*$$

et de les compter suivant la statistique $|w|_a + |w|_c = n$ et $|w|_b + |w|_c = m$. Une expression rationnelle non ambiguë pour le langage L découle de la relation suivante :

$$\begin{aligned} L &= \epsilon + a^+ + b^+ + a^+ cL + b^+ cL + cL \\ L &= (a^+ c + b^+ c + c)^* (\epsilon + a^+ + b^+) \end{aligned}$$

D'après la théorie générale des langages rationnels [26][38], on peut passer à la série génératrice rationnelle :

$$\begin{aligned} G(X,Y) &= \frac{1 + \frac{X}{1-X} + \frac{Y}{1-Y}}{1 - \frac{X^2 Y}{1-X} - \frac{XY^2}{1-Y} - XY} \\ &= \frac{1-XY}{1-X-Y+X^2Y^2} \end{aligned}$$

Le coefficient de la série en $X^n Y^m$ donne le nombre recherché de chemins.
En posant, $Z = X + Y - X^2Y^2$, on a :

$$\begin{aligned} \frac{1}{1-X-Y+X^2Y^2} &= \sum_{k \geq 0} Z^k \\ &= \sum_{k \geq 0} \sum_{k_1+k_2+k_3=k} (-1)^{k_3} \frac{k!}{k_1! k_2! k_3!} X^{k_1+2k_3} Y^{k_2+2k_3} \end{aligned}$$

$$\begin{aligned} G(X,Y) &= \frac{1}{1-Z} - \frac{XY}{1-Z} \\ &= \sum_{k \geq 0} \sum_{k_1+k_2+k_3=k} (-1)^{k_3} \frac{k!}{k_1! k_2! k_3!} X^{k_1+2k_3} Y^{k_2+2k_3} - \\ &\quad \sum_{k \geq 0} \sum_{k_1+k_2+k_3=k} (-1)^{k_3} \frac{k!}{k_1! k_2! k_3!} X^{k_1+2k_3+1} Y^{k_2+2k_3+1} \end{aligned}$$

En posant :

$$\delta_{m,n} = \sum_{0 \leq k \leq inf(\frac{m}{2}, \frac{n}{2})} (-1)^k \frac{(m+n-3k)!}{(n-2k)!(m-2k)!k!}$$

on obtient :

$$\begin{aligned} G(X,Y) &= \sum_{m\geq 0} \delta_{m,0} X^m + \sum_{n\geq 0} \delta_{0,n} Y^n + \sum_{m,n\geq 1} (\delta_{m,n} - \delta_{m-1,n-1}) X^m Y^n \\ &= \sum_{m\geq 0} X^m + \sum_{n\geq 0} Y^n + \sum_{m,n\geq 1} (\delta_{m,n} - \delta_{m-1,n-1}) X^m Y^n \end{aligned}$$

Finalement, si $C_{m,n}$ est ne nombre d'alignements possibles entre 2 séquences de longueur respective $m$ et $n$, on a :

$$C_{m,n} = \begin{cases} 1 & \text{si } m.n = 0 \\ \delta_{m,n} - \delta_{m-1,n-1} & \text{si } m.n \geq 1 \end{cases}$$

### 2.6.2 Nombre d'alignements locaux

On se ramène au cas de l'alignement global, en remarquant que tout alignement local peut être vu comme un alignement global de deux sous-séquences. Cependant, comme nous l'avons vu dans la section 2.5, un alignement local commence et se termine par des déplacements diagonaux. Fixer les sous-séquences revient à imposer les deux appariements aux extrémités de l'alignement local.

Il en découle que le nombre d'alignements locaux s'exprime à partir des nombres d'alignements globaux pour chacun des couples de sous-séquences des séquences initiales :

$$N_{local} = \sum_{\substack{[k,l] \text{ sous-séq} \\ [i,j] \text{ sous-séq}}} N_{global}(l-k+1-2, j-i+1-2) \tag{2.31}$$

## 2.7 Politiques de remontée

Le score obtenu par un algorithme de programmation dynamique (Smith & Waterman, Needleman & Wunsch, Miller & Huang...) nécessite une seule passe de l'algorithme, souvent appelée **phase de descente**[2]. Seul le score est calculé, mais on peut connaître à la volée la position de la fin de l'alignement.

Par la maintenance d'une variable $Début(i,j)$ représentant le début de l'alignement optimal se terminant en $(i,j)$, on peut récupérer en une seule passe, la position de l'alignement, c'est-à-dire le début et la fin de l'alignement. Cependant l'alignement lui-même nécessite une **phase de remontée** (en anglais *backward, backtracking*), du type de celle de Miller & Myers [93] (Cf. section 1.5.1). Lors de la comparaison d'une séquence contre une banque, ou d'une banque contre une autre, la phase de remontée n'est effectuée que pour les meilleurs scores.

Comme nous l'avons déjà évoqué, il peut y avoir plusieurs alignements de même score se terminant au même endroit (Cf. section 1.2.2, figure 1.5). La question est maintenant de générer tous ces alignements de score optimal se terminant en un point donné.

Lors de la phase de remontée, plusieurs chemins sont possibles lorsque deux valeurs sont égales au maximum dans la recurrence principale de l'algorithme. Il faut donc faire un choix, et ce choix peut faire apparaître des régions de similarité différentes. Le nombre d'alignements possibles dépend du nombre de fois que ce choix intervient et du nombre d'alternatives pour chacun de ces choix. Dans l'exemple suivant, trois choix sont nécessaires : on obtient 12 alignements possibles (Cf. figure 2.18).

2. On parle de *forward pass* en anglais.

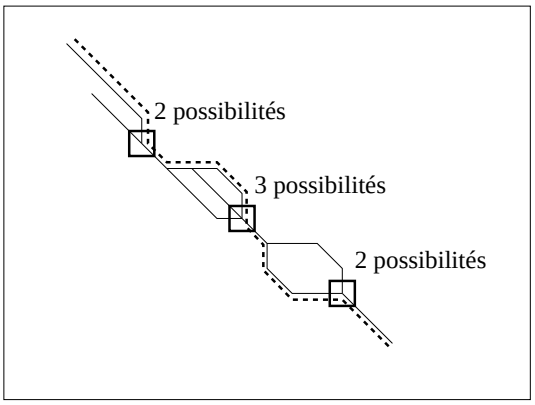



FIG. 2.18 - **Le nombre d'alignements dépend du nombre d'alternatives lors de la phase de remontée.** *Ici, le nombre total d'alignements optimaux est* $2 \times 3 \times 2 = 12$. *L'alignement représenté en pointillé est obtenu par la politique de remontée numéro* 6 *(Cf. tableau 2.2).*

Cependant, on peut par exemple imposer des choix automatiques. Dans le cas où l'on doit choisir entre une *diagonale* et une *verticale*, on peut préférer remonter par la **diagonale**. Toute règle de remontée sera appelée une **politique de remontée**.

Il y a 6 politiques de remontée. Elles sont décrites par les règles de priorité lors de la phase de remontée :

| | priorités | |
|---|---|---|
| politique | numéro 1 | numéro 2 |
| 1 | diagonale | verticale |
| 2 | diagonale | horizontale |
| 3 | verticale | diagonale |
| 4 | verticale | horizontale |
| 5 | horizontale | diagonale |
| 6 | horizontale | verticale |

TAB. 2.2 - **Les différentes politiques de remontée pour l'algorithme de programmation dynamique.**

Supposons que l'on choisisse la politique numéro 6. Lors de la phase de remontée, parmi l'ensemble des chemin ayant permis d'atteindre le maximum, on choisira d'abord l'horizontale. Si l'horizontale a un score non optimal, on choisira la verticale. Si la verticale ne permet pas d'atteindre l'optimum, la diagonale constituera le chemin retenu (Cf figure 2.18).

Le choix d'une politique de remontée influe sur les régions de similarité mises en évidence. Lorsqu'on change de politique de remontée, l'alignement trouvé peut ne pas commencer au même endroit. Il peut être difficile de déterminer le «*bon*» alignement. La politique de remontée est aussi importante que le début d'un alignement.

On peut alors s'intéresser à l'ensemble des alignements optimaux. Mais, le nombre d'alignements peut être élevé. Si on génère cet ensemble, les résultats sont difficiles à dépouiller, et le biologiste

préfère souvent modifier localement un alignement, en fonction de sa connaissance des séquences présentes.

Nous avons déjà vu que certaines méthodes permettent de favoriser des alignements avec un grand nombre d'identités : l'algorithme de Huang favorise les alignements avec des appariements exacts consécutifs. Dans certains cas, en changeant la politique de remontée et les paramètres, on peut retrouver grâce à l'algorithme classique de Smith & Waterman, les alignements donnés par l'algorithme de Huang. Reprenons le premier exemple de Huang :

```
hemoglobin alpha chain - Arabian camel        (PIR1 : HACMA)
hemoglobin beta  chain - moustached tamarin   (PIR1 : HBMKM)
```

Dans son article, Huang choisissait la matrice PAM250. Si on prend la matrice DAYHOFF, on peut retrouver les alignements de Huang parmi les différents alignements donnés par Smith & Waterman, à condition de choisir la bonne politique de remontée (Cf. figure 2.19). L'alignement donné par l'algorithme de Huang est très proche d'un alignement optimal.

```
       DAYHOFF gapo=7 gape=2 politique=1,2,3

 1 V-LSSKDKTNVKTAFGKIGGHAAEYGAEALERMFLGFPTTKTYFPHF-DL
   |-|    |  | |  || --   | | ||| |     | |   |  |-||
 1 VHLTGEEKSAVTTLWGKV--NVEEVGGEALGRLLVVYPWTQRFFDSFGDL

50     .    :    .    :    .    :    .    :    .    :
49 SH-----GSAQVKAHGKKVGDALTKAADHLDDLPSALSALSDLHAHKLRV
   | -----    ||||||||  |      ||| |      || ||  || |
49 SSPDAVMNNPKVKAHGKKVLGAFSDGLAHLDNLKGTFAQLSELHCDKLHV

100    .    :    .    :    .    :    .    :    .
94 DPVNFKLLSHCLLVTVAAHHPGDFTPSVHASLDKFLANVSTVLTSKYR
   || || ||  |    | |   ||| | |   |  | |   |  ||
99 DPENFRLLGNVLVCVLAHHFGKEFTPQVQAAYQKVVAGVANALAHKYH

       DAYHOFF gapo=7 gape=2 politique=4,5,6

 1 V-LSSKDKTNVKTAFGKIGGHAAEYGAEALERMFLGFPTTKTYFPHF-DL
   |-|    |  | |  || --   | | ||| |     | |   |  |-||
 1 VHLTGEEKSAVTTLWGKV--NVEEVGGEALGRLLVVYPWTQRFFDSFGDL

50     .    :    .    :    .    :    .    :    .    :
49 SH-----GSAQVKAHGKKVGDALTKAADHLDDLPSALSALSDLHAHKLRV
   | -----    ||||||||  |      ||| |      || ||  || |
49 SSPDAVMNNPKVKAHGKKVLGAFSDGLAHLDNLKGTFAQLSELHCDKLHV

100    .    :    .    :    .    :    .    :    .
94 DPVNFKLLSHCLLVTVAAHHPG-DFTPSVHASLDKFLANVSTVLTSKYR
   || || ||  |- || | ||| |- ||| | |   |  | |   |  ||
99 DPENFRLLGN-VLVCVLAHHFGKEFTPQVQAAYQKVVAGVANALAHKYH
```

FIG. 2.19 - **Politiques de remontée et alignement :** *Les politiques de remontée 1, 2 et 3 ne favorisent pas les appariements exacts. Par contre en utilisant les autres politiques, on retombe sur les alignements trouvés par Huang grâce à son algorithme favorisant les alignements ayant des appariements exacts concentrés.*

Le problème soulevé ici est finalement la recherche de l'alignement, le plus vraissemblable du point de vue biologique, parmi les alignements optimaux et sous-optimaux. L'algorithme de Huang permet de choisir un alignement qui comporte des zones de forte identité, mais d'autres solutions sont envisageables. Par exemple, si on modifie la matrice de substitution en augmentant les éléments diagonaux, l'algorithme habituel de Smith & Waterman préfère les identités, et on peut trouver des alignements qui ont plus d'identités que ceux trouvés avec les matrices habituelles. Une étude sur la dépendance de l'alignement en fonction des paramètres pourrait permettre de différencier les alignements optimaux.

## 2.8 Conclusion

Les algorithmes de Needleman & Wunsch et de Smith & Waterman peuvent être accélérés, grâce à la prise en compte de remarques subtiles. Pour l'alignement global, le nombre d'opérations peut être diminué en donnant une autre signification au score courant. Cependant, il n'a pas été possible de reporter ce gain sur l'algorithme de Smith & Waterman.

Dans ce cas, l'implémentation en parallèle grâce aux instructions VIS, a permis de mettre en évidence certaines propriétés qui sont applicables au cas du pattern matching. Au coeur de l'algorithme, ce sont les tests qui coûtent cher. Or ces opérations ne sont pas toutes constamment indispensables. Le gain dû à la nouvelle implémentation est important.

Les deux algorithmes d'alignements de séquences (local et global) sont très utilisés, et on affirme souvent sans argument précis que le score Smith & Waterman est une optimisation du score Needleman & Wunsch sur l'ensemble des sous-séquences. Les relations entre les deux algorithmes ont été analysées ici avec soin, et confirment le résultat.

Le travail présenté sur le dénombrement donne une expression exacte du nombre d'alignements globaux entre deux séquences de longueurs fixes. Cette expression n'est pas simple, mais le nombre d'alignements peut être calculé en $O(inf(\frac{m}{2}, \frac{n}{2}))$, sans avoir à implémenter une récurrence sur deux indices.

Enfin, la prise en compte de la politique de remontée pose un problème plus ambitieux : trouver l'alignement, biologiquement le plus significatif, parmi l'ensemble des alignements optimaux.

# Chapitre 3

# L'algèbre $(\max, +)$ au secours de la Programmation Dynamique

*Puisque les algorithmes de Smith & Waterman ou de Needleman & Wunsch ne font apparaître que des additions et des maximisations entre plusieurs valeurs, on peut les formuler dans l'algèbre* $(\max, +)$. *Parler d'algèbre est un abus de langage, puisqu'il s'agit en fait d'un semi-anneau. L'utilisation de cette structure algébrique pour les problèmes d'optimisation a été bien étudiée dans [10].*

*Considérons pour fixer les idées le cas le plus simple d'une fonction linéaire de pénalisation des insertions/délétions. Lors de la comparaison de deux séquences* $A[1,n]$ *et* $B[1,m]$, *de longueur respective* $n$ *et* $m$, *on itère* $n$ *fois la multiplication dans l'algèbre* $(\max, +)$ *d'un vecteur ligne* $X_l$ *de taille* $m$ *par une matrice triangulaire supérieure* $C_l$ *:*

$$X_l = X_{l-1} \otimes C_l$$

*L'ensemble des matrices* $\{C_l\}_l$ *est très réduit : il y a autant de matrices que de lettres dans l'alphabet utilisé (*4 *dans le cas de l'ADN, et* 20 *dans le cas des protéines). Chacune de ces matrices ne dépend que de la séquence* $B[1,m]$. *Le semi-groupe engendré par l'ensemble des matrices* $\{C_a, a \in \mathcal{A}\}$ *est fini ou projectivement fini. Il s'en suit que l'ensemble des produits consécutifs des matrices* $C_l$ *peut être précalculé et qu'un automate peut être construit pour répondre au problème de l'alignement de séquences par programmation dynamique.*

*Lors d'une comparaison exhaustive de la séquence* $B[1,m]$ *contre toute une banque de données, l'automate n'est construit qu'une seule fois. L'algorithme permettant le calcul du score, après construction de l'automate, se réduit à un simple parcours de l'automate et devient* **linéaire en temps**.

## 3.1 Rappel sur le semi-anneau $(Max, +)$

Notre but ici n'est pas l'étude de la structure algébrique $(max, +)$. Cependant un rappel sur cette structure permettra au lecteur de se familiariser avec les notations. Il s'agit juste de quelques définitions et de quelques théorèmes qui nous sont apparus utiles pour la suite.

**Définition 3.1.1** *On appelle semi-anneau* $\mathcal{K}$ *un ensemble doté de deux opérations internes notées* $\oplus$ *et* $\otimes$ *tel que*

– *l'opération* $\oplus$ *est associative, commutative et a un élément neutre noté* $\varepsilon$,

– *l'opération* $\otimes$ *définit un monoïde sur* $\mathcal{K}$ *dont l'élément neutre est noté* $e$. *De plus* $\otimes$ *est distributive par rapport à l'addition* $\oplus$, *et* $\varepsilon$ *est absorbant, i.e.*

$$\varepsilon \otimes a = a \otimes \varepsilon = \varepsilon \quad \forall a \in \mathcal{K}$$

*Nous dirons que le semi-anneau est :*

– *idempotent si la première opération l'est :* $a \oplus a = a, \forall a \in \mathcal{K}$

– *commutatif si le monoïde l'est.*

De manière analogue au cas des semi-anneaux, on définit les semi-algèbres en renonçant à exiger que l'addition soit une loi de groupe dans les axiomes de structure des algèbres.
On dit qu'une semi-algèbre est idempotente si l'addition l'est.

**Définition 3.1.2** *: La structure* $\mathbb{R}_{max}$
*On appellera* $\mathbb{R}_{max}$ *l'ensemble* $\mathbb{R} \cup \{-\infty\}$ *muni des opérations* max *et* $+$, *notées respectivement* $\oplus$ *et* $\otimes$.

**Exemples :** Les opérations élémentaires peuvent surprendre :

$$1 \oplus 3 = 3$$
$$1 \otimes 3 = 4$$

Le résultat suivant est clair.

**Théorème 3.1.1** *La structure algébrique* $\mathbb{R}_{max}$ *est un semi-anneau idempotent commutatif. L'élément neutre pour* $\oplus$ *est* $\varepsilon = -\infty$, *l'élément neutre pour la deuxième opération est* $e = 0$.

Dans le semi-anneau $\mathbb{R}_{max}$, on peut parler de matrices. On définit les deux opérations suivantes :

– Etant données deux matrices de taille identique, $A \in \mathbb{R}_{max}^{m \times k}$ et $B \in \mathbb{R}_{max}^{m \times k}$, l'addition de ces deux matrices est la matrice de $\mathbb{R}_{max}^{m \times k}$, dont les éléments sont donnés par :

$$(A \oplus B)_{i,j} \quad = \quad A_{i,j} \oplus B_{i,j} \quad \forall (i,j) \in [1,m] \times [1,k].$$

– Etant données deux matrices $A$ et $C$, $A \in \mathbb{R}_{max}^{m \times k}$ et $C \in \mathbb{R}_{max}^{k \times p}$, la matrice $A \otimes C$ est la matrice de $\mathbb{R}_{max}^{m \times p}$ dont les élements valent :

$$(A \otimes C)_{i,j} \quad = \quad \bigoplus_{l=1}^{k} A_{i,l} \otimes C_{l,j} \quad \forall (i,j) \in [1,m] \times [1,p].$$

**Définition 3.1.3** *: La structure* $(\mathbb{R}_{max})^{n \times n}$
*Soit* $(\mathbb{R}_{max})^{n \times n}$ *l'ensemble des matrices de taille* $n \times n$ *à coefficients dans* $\mathbb{R}_{max}$ *doté des opérations internes suivantes :*

– *l'addition terme à terme notée* $\oplus$,

– *la multiplication notée* $\otimes$

$$(A \otimes B)_{i,j} = \bigoplus_{k=1}^{n} A_{i,k} \otimes B_{k,j}$$

*et de l'opération externe :*

– $\forall a \in \mathbb{R}_{max}, \forall A \in (\mathbb{R}_{max})^{n\times n}$, $a \otimes A$ *est la matrice dont les coefficients sont* $(a \otimes A_{i,j})$.

*L'ensemble* $\mathbb{R}_{max}^{n\times n}$ *est une semi-algèbre idempotente avec :*

– *la matrice nulle dont tous les coefficients valent* $\varepsilon$,

– *la matrice identité dont les coefficients diagonaux valent* $e$ *et les autres coefficients* $\varepsilon$.

$$I = \begin{pmatrix} e & \varepsilon & \dots & \varepsilon \\ \varepsilon & e & \ddots & \vdots \\ \vdots & \ddots & \ddots & \varepsilon \\ \varepsilon & \dots & \varepsilon & e \end{pmatrix}$$

**Exemple de calcul matriciel dans** $\mathbb{R}_{max}^{2\times 2}$ **:**

$$\begin{pmatrix} 3 & 2 \\ 4 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 2 \\ 1 & 5 \end{pmatrix} = \begin{pmatrix} 3 & 7 \\ 4 & 6 \end{pmatrix}$$

### 3.1.1 Matrices et graphes de précédence

A partir de toute matrice de $(\mathbb{R}_{max})^{n\times n}$ on peut construire un graphe valué orienté sur $n$ sommets. Les sommets sont numérotés de 1 à $n$, et les arcs du graphe sont définis par :

– un arc existe entre les deux sommets $i$ et $j$ si l'élément $A_{i,j}$ n'est pas l'élément $\varepsilon$,

– le poids de l'arc allant de $i$ à $j$ est exactement $A_{i,j}$ si cette dernière valeur n'est pas $\varepsilon$.

Le poids d'un chemin $(i_1, \dots, i_k)$ allant de $i$ à $j$ (i.e. $i_1 = i$ et $i_k = j$) est par définition la somme des poids des arcs le long de ce chemin :

$$A_{i_1 i_2} + A_{i_2 i_3} + \cdots + A_{i_{k-1} i_k}.$$

Nous appellerons poids maximum des chemins allant de $i$ à $j$, la borne supérieure de ces quantités, lorsque $(i_1, \dots, i_k)$ parcourt l'ensemble des chemins de longueur au moins égale à 1, allant de $i$ à $j$. S'il n'y a aucun chemin allant de $i$ à $j$, on conviendra que le poids maximum est $-\infty$.

Notons que le poids maximum peut valoir $+\infty$. C'est le cas s'il existe un chemin de $i$ vers $j$ passant par un sommet $k$ appartenant à un circuit de coût positif. En effet, pour aller de $i$ à $j$, on peut aller d'abord en $k$, passer un nombre quelconque de fois par le circuit de coût positif, puis aller en $j$.

Le poids du chemin de poids maximum pour aller de $i$ à $j$ en un seul arc est forcement $A_{i,j}$. Le poids du chemin pour aller de $i$ à $j$ en passant par $k$ est $A_{i,k} + A_{k,j}$. Le chemin de coût maximum pour aller de $i$ à $j$ en utilisant exactement 2 arcs a un poids égal à :

$$\max_{k=1}^{n}(A_{i,k} + A_{k,j}) = \bigoplus_{k=1}^{n} A_{i,k} \otimes A_{k,j} = (A^2)_{i,j}$$

où $A^2$ est le carré au sens $(max, +)$ de la matrice $A$.

Si on cherche le maximum des poids des chemins allant de $i$ à $j$ passant par $k$ sommets intermédiaires, le résultat sera donné par $(A^{k+1})_{i,j}$. On montre ainsi que la borne supérieure des poids des chemins allant de $i$ à $j$ est donnée par :

$$(A \oplus A^2 \oplus A^3 \oplus \cdots \oplus A^n \oplus \cdots)_{i,j}$$

Si nous définissons les opérations notées $^*$ et $^+$ :

$$\begin{aligned} A^* &= e \oplus A \oplus A^2 \oplus \cdots \oplus A^n \oplus \cdots \\ A^+ &= \quad\; A \oplus A^2 \oplus \cdots \oplus A^n \oplus \cdots = A \otimes A^* \end{aligned}$$

la borne supérieure du poids du chemin de poids maximum entre les sommets $i$ et $j$ est donnée par l'élément $(i, j)$ de la matrice $A^+$.

### 3.1.2 Système de la forme $Ax \oplus b = x$

L'algorithme de Smith & Waterman peut-être vu comme un problème de coût maximum dans un graphe orienté valué. Le paragraphe précédent fait le lien entre l'algèbre $(max, +)$ et ce type de problèmes. Exprimer l'algorithme de Smith & Waterman dans cette algèbre exige de pouvoir résoudre des systèmes linéaires. Les deux théorèmes suivants sont utiles pour la suite.

**Théorème 3.1.2** *S'il n'y a que des circuits de poids négatifs ou nuls dans le graphe de précédence associé à la matrice $A$, il existe une solution à l'équation $Ax \oplus b = x$ qui est donnée par $x = A^*b$. Si en outre il n'y a pas de circuit de poids nul, la solution est unique.*

**Théorème 3.1.3** *Si le graphe de précédence n'a pas de circuit de poids positif, alors $A^* = e \oplus A \oplus A^2 \oplus A^3 \ldots \oplus A^{n-1}$ où $n$ est la dimension de la matrice $A$.*

> **Démonstration** : Soit $p$ un chemin de $i$ à $j$. Si $p$ est de longueur au moins égale à $n$, le chemin $p$ passe au moins deux fois par le même sommet. On peut donc factoriser le chemin : $p = p_1 c_1 p_2 \cdots p_k c_k p_{k+1}$ où les $c_i$ sont des circuits élémentaires ou triviaux, et où $p_1 p_2 \cdots p_{k+1}$ est un chemin sans circuit de $i$ à $j$.
>
> S'il n'y a pas de circuit de poids positif, le poids du chemin est inférieur au poids du chemin sans circuit. Donc le maximum dans l'étoile $A^*$ est atteint pour un chemin sans circuit de longueur au maximum égale à $n-1$. On a donc $A^* = e \oplus A \oplus A^2 \oplus A^3 \oplus \ldots \oplus A^{n-1}$.
> □

On peut montrer réciproquement que s'il existe un circuit de poids strictement positif, $A^*$ ne converge pas dans $(\mathbb{R}_{max})^{n \times n}$.
En effet, soit $c$ un circuit (disons de longueur $k$) de poids strictement positif passant pas $i$. On a :

$$(A^{nk})_{ii} \geq p(c)^n \longrightarrow +\infty$$

où $p(c)$ est le poids du chemin. Ceci montre que l'étoile $A^*$ ne converge pas dans $(\mathbb{R}_{max})^{n \times n}$.

## 3.2 Méthode des «matrices de transfert» pour le calcul du score Needleman & Wunsch

Considérons 2 séquences à comparer $A$ et $B$ de longueurs respectives $l_A$ et $l_B$. Contrairement aux chapitres précédents, nous n'allons plus faire jouer aux séquences $A$ et $B$ des rôles similaires. Même si le résultat de l'algorithme est identique, nous distinguerons les deux séquences.

Nous allons pré-effectuer une partie du calcul, et cette première phase dépend de l'une des deux séquences. Par convention, cette première phase dépend de la séquence $B$. Ce précalcul est valable pour les alignements de la séquence $B$ avec n'importe quelle autre séquence $A$. Cette première phase n'est effectuée qu'une seule fois lorsqu'on cherche à aligner une séquence avec toutes les séquences d'une banque. Nous appellerons la séquence $B$, la *séquence requête*, puisqu'on cherche dans la banque de séquences celles qui ressemblent à $B$. De manière symétrique, les séquences de la banque de données seront appelées *séquences cibles*.

Pour commencer, on simplifie l'algorithme en prenant une fonction linéaire de pénalisation des insertions/délétions : $g(k) = -\delta.k$. Notons $N_{i,j}$ la valeur donnée par l'algorithme NW, en position $i$ et $j$. En posant $\sigma_{i,j} = S(A[i], B[j])$ la valeur de la matrice de substitution pour les lettres $A[i]$ et $B[j]$, on a la récurrence suivante :

$$\begin{array}{rcl} N_{i,j} & = & \max\Big(N_{i-1,j-1} + \sigma_{i,j} \quad , \quad N_{i-1,j} - \delta \quad , \quad N_{i,j-1} - \delta\Big) \\ & = & N_{i-1,j-1} \otimes \sigma_{i,j} \quad \oplus \quad \delta^{-1} \otimes N_{i-1,j} \quad \oplus \quad \delta^{-1} \otimes N_{i,j-1} \end{array} \tag{3.1}$$

Notons que les opérations $\oplus$ et $\otimes$ ont les priorités usuelles. Les conditions initiales de l'algorithme NW sont données par :

$$\left\{ \begin{array}{lcll} N(0,0) & = & 0 & \\ N(i,0) & = & g(i) = -\delta \times i & \forall i \in [1, l_A] \\ N(0,j) & = & g(j) = -\delta \times j & \forall j \in [1, l_B] \end{array} \right. \tag{3.2}$$

Soit $X_n$ un vecteur *ligne* de taille $l_B + 1$ correspondant à la $n^{ième}$ ligne du tableau habituel de l'algorithme NW,

$$X_n = \Big( \begin{array}{ccccc} N(n,0) & N(n,1) & N(n,2) & \ldots & N(n,l_B) \end{array} \Big) \tag{3.3}$$

On peut écrire la récurrence à l'aide des vecteurs $X_n$ :

$$X_n \quad = \quad X_{n-1} \otimes A_n \quad \oplus \quad X_n \otimes B_n \tag{3.4}$$

avec

$$B_n \quad = \quad \begin{pmatrix} . & \delta^{-1} & . & . & . \\ . & . & \delta^{-1} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \delta^{-1} \\ . & . & . & . & . \end{pmatrix}$$

$$A_n \quad = \quad \begin{pmatrix} \delta^{-1} & \sigma_{1,n} & . & \ldots & . \\ . & \delta^{-1} & \sigma_{2,n} & \ldots & . \\ . & . & \delta^{-1} & \ddots & . \\ \vdots & \vdots & \vdots & \ddots & \sigma_{l_B,n} \\ . & . & . & . & \delta^{-1} \end{pmatrix}$$

où

- $\sigma_{i,n}$ est la valeur de la matrice de substitution pour les lettres $B[i]$ et $A[n]$,
- les valeurs non spécifiées valent l'élément absorbant du semi-anneau $(max, +)$, c'est-à-dire $-\infty$.

On remarque que les matrices $B_n$ sont indépendantes de $n$, et on parlera simplement de $B$. Puisque la matrice $B$ est une surdiagonale, le graphe associé n'a aucun circuit. D'àprès le théorème 3.1.3, la suite $\left(\bigoplus_{i=0}^{k} B^i\right)_{k\in\mathbb{N}}$ converge. En fait, il est trivial de montrer que toutes les puissances de $B$, $\left(B^i\right)_{i\geq l_B}$ sont nulles. On a :

$$B^* = \bigoplus_{i=0}^{\infty} B^i = \begin{pmatrix} e & \delta^{-1} & \delta^{-2} & \delta^{-3} & \dots & \delta^{-l_B} \\ . & e & \delta^{-1} & \delta^{-2} & \dots & \delta^{-l_B+1} \\ . & . & e & \delta^{-1} & \dots & \delta^{-l_B+2} \\ . & . & . & \ddots & \ddots & \vdots \\ . & . & . & . & e & \delta^{-1} \\ . & . & . & . & . & e \end{pmatrix} \tag{3.5}$$

Quel que soit le vecteur ligne $b$, $b\otimes B^*$ est solution de l'équation linéaire $X = XB\oplus b$. En particulier, on a :

$$\begin{aligned} X_n &= X_nB \ \oplus \ X_{n-1}A_n \\ &= X_{n-1}A_nB^* \\ &= X_{n-1}\otimes A_n\otimes B^* \end{aligned}$$

En posant $C_n = A_n \otimes B^*$, on obtient :

$$X_n = X_{n-1}\otimes C_n \tag{3.6}$$

avec

$$C_n = \begin{pmatrix} \delta^{-1} & & & & \\ . & \delta^{-1} & & C_{i,j} & \\ . & . & \delta^{-1} & & \\ . & . & . & & \\ . & . & . & & \delta^{-1} \end{pmatrix}$$

$$\begin{cases} si\ i<j & C_{i,j} = \begin{cases} \left(\delta^{-2}\oplus\sigma(i,n)\right) \\ \otimes\delta^{-(|i-j|-1)} \end{cases} & = \begin{cases} \max\left(-2\delta, \sigma(i,n)\right) \\ -(|i-j|-1).\delta \end{cases} \\ si\ i>j & C_{i,j} = \varepsilon & = -\infty \\ si\ i=j & C_{i,j} = \delta^{-1} & = -\delta \end{cases}$$

Ce calcul se résume au résultat suivant.

**Théorème 3.2.1** *Posons $X_0 = (0, \delta^{-1}, \delta^{-2}, \dots, \delta^{-l_B})$. Le score Needleman & Wunsch pour les séquences $A$ et $B$ coïncide avec la dernière composante du vecteur*

$$X_0 \otimes C_1 \otimes C_2 \otimes \cdots \otimes C_{l_A}.$$

**Démonstration** : $X_0$ est égal à la première ligne du tableau habituel de l'algorithme de Needleman & Wunsch (Cf. éq. 3.2 et 3.3). De plus, nous venons de montrer que $X_n = X_{n-1} \otimes C_n$. Chaque multiplication de matrice correspond au passage d'une ligne à une autre dans la matrice de score de Needleman & Wunsch.

Puisque le score Needleman & Wunsch se lit dans la dernière ligne du tableau habituel, on le trouve dans la dernière composante du vecteur correspondant à la dernière ligne.

□

**Remarque :** La matrice se simplifie si on prend en compte la contrainte supplémentaire :

$$\delta^{-2} \leq \min_{a,b}(\sigma(a,b))$$

On peut donner une interprétation à cette contrainte. Lors de l'alignement de deux séquences, on préfèrera toujours une substitution à deux insertions/délétions. Ceci correspond à la minimisation du nombre d'événements de mutations.
La matrice a la même forme mais les éléments deviennent :

$$\begin{cases} si\ i < j & C_{i,j} = \sigma(i,n) \otimes \delta^{-(|i-j|-1)} = \sigma(i,n) - (|i-j|-1).\delta \\ si\ i > j & C_{i,j} = \varepsilon = -\infty \\ si\ i = j & C_{i,j} = \delta^{-1} = -\delta \end{cases}$$

L'expression (3.6) implique, qu'à une étape donnée $l$, le score $X_{l,i}$ ne dépend que des valeurs $(X_{l-1,k})_{k \leq i}$; ce qui est tout à fait normal, puisque tout chemin qui arrive en $(l,i)$, passe par l'une des cases $(X_{l-1,k})_{k \leq i}$.

## 3.3 Le semi-groupe des matrices $C_n$

Pour une séquence donnée $B[1, l_B]$, on a 4 ou 20 matrices $C_n$, suivant qu'il s'agit d'une séquence d'ADN ou d'une protéine. La matrice $C_i$ correspond à la matrice associée à la lettre de l'alphabet $A[i]$. On dispose d'un petit nombre de matrices $\{C_a, a \in \Sigma\}$, où $\Sigma$ est l'alphabet utilisé.

Lorsqu'on compare la séquence $B$ à une séquence quelconque $A$, on effectue $l_A$ produits de matrices : $\otimes_{l=1}^{l_A} C_{A[l]}$. Le score NW se lit dans la dernière colonne du vecteur ligne obtenu.

On cherche alors à construire l'ensemble des vecteurs possibles pour $X_n$. Si on connaît la structure de cet ensemble, on peut accélérer le calcul du score.

Nous allons montrer que, pour une séquence $B$ fixée, l'ensemble des valeurs possibles de $X_n$ ne possède qu'un nombre fini d'éléments non proportionnels. Pour cela, nous sommes amenés à définir les espaces projectifs.

**Définition 3.3.1** *L'espace projectif de $\mathbb{R}_{max}^n$ noté $\mathbb{PR}_{max}^n$, de dimension $n-1$, est défini comme étant le quotient de $\mathbb{R}_{max}^n$ par la relation d'équivalence $\sim$ :*

$$\begin{gathered} pour\ \ u \in \mathbb{R}_{\max}^n\ \ et\ \ v \in \mathbb{R}_{\max}^n \\ u \sim v \Longleftrightarrow \exists \lambda \in \mathbb{R}_{max} \backslash \{\varepsilon\},\quad u = \lambda v \end{gathered}$$

**Définition 3.3.2** *On définit de même l'espace projectif* $\mathbb{PR}_{max}^{n\times n}$ *comme étant le quotient de l'espace des matrices carrées* $(max,+)$ *de dimension n par la relation d'équivalence :*

$$\begin{array}{c} pour\ \ U \in \mathbb{R}_{\max}^{n\times n}\ \ et\ \ V \in \mathbb{R}_{\max}^{n\times n} \\ U \sim V \Longleftrightarrow \exists \lambda \in \mathbb{R}_{max}\backslash\{\varepsilon\},\quad U = \lambda V \end{array}$$

**Définition 3.3.3** *On dira qu'un ensemble* $S \in \mathbb{R}_{max}^{n\times n}$ *est* **projectivement fini** *si l'ensemble quotient* $S/\sim$ *est fini, c'est-à-dire s'il n'existe qu'un nombre fini d'éléments deux à deux non proportionnels.*

On notera $\langle C_a, a \in \Sigma\rangle$ le semi-groupe multiplicatif, au sens $(max,+)$, engendré par les matrices $C_a, a \in \Sigma$. C'est l'ensemble des matrices de la forme $C_{a_1} \otimes C_{a_2} \otimes \cdots \otimes C_{a_k}$, avec $a_1 a_2 \cdots a_k \in \Sigma^*$.

**Théorème 3.3.1** *Le semi-groupe* $\langle C_a, a \in \Sigma\rangle$ *est* projectivement fini.

**Démonstration** : Définissons $\tilde{C}_a = \delta C_a, \forall a \in \Sigma$. Posons $n = l_B + 1$ la dimension des matrices du semi-groupe.

Définissons un automate.

- L'ensemble des états de l'automate est l'ensemble des coordonnées d'un vecteur ligne $\mathcal{S} = \{0, 1, 2, \ldots, l_B\}$.
- Il y a autant de transitions de $i$ vers $j$ qu'il existe de lettres $a \in \Sigma$ telles que le graphe de précédence associé à la matrice $\tilde{C}_a$ contient l'arc $(i \longrightarrow j)$. Les transitions seront notées $(i \xrightarrow{a} j)$ où $a$ désigne une lettre telle que $(\tilde{C}_a)_{i,j} \neq \varepsilon$.
- De plus on associe un poids à chacune de ces transitions. La transition $(i \xrightarrow{a} j)$ aura un poids égal à $(\tilde{C}_a)_{i,j}$.

Cet automate peut être vu comme un graphe possédant éventuellement plusieurs arcs allant de $i$ à $j$. Puisque les matrices sont triangulaires supérieures, les arcs $(i \longrightarrow j)$ sont tels que $i \leq j$. Les seuls circuits de ce graphe sont les boucles simples allant de $k$ à $k$, de poids nul.

Soit $M$ une matrice du semi-groupe $\langle \tilde{C}_a, a \in \Sigma\rangle$. $M$ peut s'écrire $M = \bigotimes_{i=1}^{l} (\tilde{C}_{a_i})$. Puisque toutes les matrices $\tilde{C}_a, a \in \Sigma$, ont des $e$ sur la diagonale, les éléments diagonaux de $M$ sont aussi égaux à $e$. $M_{i,j}$ représente le maximum des coûts des chemins de label $a_1 a_2 a_3 ... a_l$ allant de $i$ à $j$.

Considérons dans le graphe précédent, un chemin quelconque allant de $i$ à $j$. Posons $i = i_1, i_2, i_3 \ldots i_k = j$ ce chemin. Puisque les matrices sont triangulaires supérieures, la suite $(i_l)_{l=1..k}$ est croissante. Il n'y a qu'un nombre fini de chemins allant de $i$ à $j$, si on ne compte pas les boucles. Comme les boucles sont de poids nul, le poids pour un chemin quelconque allant de $i$ à $j$ ne peut prendre qu'un nombre fini de valeurs. Donc $M_{i,j}$ ne peut prendre qu'un nombre fini de valeurs, et $M$ aussi.

□

**Théorème 3.3.2** *Soit* $N(A,B)$ *le score Needleman & Wunsch entre deux séquences de longueur respective* $l_A$ *et* $l_B$*. Alors, pour une séquence B fixée, la variable*

$$N(A,B) + \delta \times l_A = \delta^{l_A} \otimes N(A,B)$$

*ne prend qu'un nombre fini de valeurs quand A parcourt* $\Sigma^*$*, où* $\Sigma$ *est l'alphabet.*

**Démonstration** : La démonstration précédente montre que le semi-groupe engendré par les matrices $\tilde{C}_a = \delta C_a, a \in \Sigma$ est fini. Or le produit consécutif des matrices $\left(\bigotimes_{i=1}^{n} \tilde{C}_{A[i]}\right)$ s'écrit :

$$\begin{aligned}\left(\bigotimes_{i=1}^{n} \tilde{C}_{A[i]}\right) &= \bigotimes_{i=1}^{n} \delta C_{A[i]} \\ &= \delta^n \otimes \left(\bigotimes_{i=1}^{n} C_{A[i]}\right)\end{aligned}$$

Si on ne regarde que l'élément $(l_B + 1)$ du vecteur $X_0 \otimes \left(\bigotimes_{i=1}^{n} \tilde{C}_{A[i]}\right)$, on a

$$\left(X_0 \bigotimes_{i=1}^{n} \tilde{C}_{A[i]}\right)_{l_B+1} = \delta^n \otimes \underbrace{\left(X_0 \bigotimes_{i=1}^{n} C_{A[i]}\right)_{l_B+1}}_{N(A,B)}$$

D'où le résultat.□

**Remarque 3.3.1** *Ce résultat ne dépend pas du caractère entier ou réel des paramètres $\sigma_{i,j}$. Ainsi, que la matrice de substitution soit réelle ou entière, le semi-groupe $\langle C_a, a \in \Sigma \rangle$ est projectivement fini.*

**Remarque 3.3.2** *On peut donner un majorant au nombre d'éléments du semi-groupe $\langle \tilde{C}_a, a \in \Sigma \rangle$.*

L'application

$$\begin{array}{rccc} \phi & : & \Sigma & \longrightarrow & \mathbb{R}_{max}^{n \times n} \\ & & a & \longrightarrow & \tilde{C}_a \end{array} \tag{3.7}$$

s'étend canoniquement en un morphisme de $\Sigma^*$ vers $\mathbb{R}_{max}^{n \times n}$ en posant $\phi(a_1 a_2 \cdots a_l) = \phi(a_1) \otimes \phi(a_2) \otimes \cdots \otimes \phi(a_l)$. Soit $\phi(w)_{i,j}$ la valeur du coefficient $(i, j)$ de la matrice associée au mot $w$ par le morphisme canonique.

Soit $F_a$ l'ensemble des valeurs différentes de $-\infty$, présentes dans la matrice $\tilde{C}_a$. $F_a$ est défini pour tout $a \in \Sigma$. Soit $F = \cup(F_a, a \in \Sigma)$.

Pour un mot $w$, $\phi(w)_{ij}$ représente le poids du chemin de label $w$ de score maximal allant de $i$ à $j$. Soit $i = i_1, i_2, i_3, \ldots, i_k = j$ ce chemin. Le coût de tout circuit étant 0, on peut toujours supposer que ce chemin est sans circuit et donc que $i_1 < i_2 < i_3 < \ldots < i_k$. Le coût de ce chemin est la somme des poids associés aux arcs. Puisque la somme est associative et commutative, le nombre de valeurs différentes pour le poids total est majoré par le nombre de tirages possibles de $k$ valeurs parmi $|F|$ avec remise et sans ordre. Il y a $K_{|F|}^{k} = C_{|F|+k-1}^{|F|-1}$ tirages possibles. Ce nombre est à son tour majoré par $|F|^k$.

Un majorant $\mathcal{N}$ des valeurs possibles pour $\phi(w)_{ij}$, quelle que soit la longueur du mot $w$ est donné par la majoration du nombre de chemins :

$$\begin{aligned}\text{Nb chemins} &= \sum_{\substack{chemin \\ i \to j}} 1 = \sum_{k=1}^{j-i} \left( \sum_{\substack{p \ chemin \ i \to j \\ |p|=k}} 1 \right) \\ &\leq \sum_{k=1}^{j-i} \left( \sum_{\substack{p \ chemin \ i \to j \\ |p|=k}} |F|^k \right) = \mathcal{N}\end{aligned}$$

Combien y a-t-il de chemins de longueur k pour aller de $i$ à $j$ ? la réponse est $C_{j-i-1}^{k-1}$. Un chemin de longueur $k$ passe par $k-1$ nœuds sans compter les 2 extrémités. Le problème revient à se demander combien y a-t-il de possibilités de choisir $k-1$ nœuds (par où le chemin passe) parmi les $j-i-1$ nœuds compris entre les sommets $i$ et $j$.

$$\begin{aligned}\mathcal{N} &= \sum_{k=1}^{j-i} |F|^k C_{j-i-1}^{k-1} \\ &= |F| \sum_{k'=0}^{j-i-1} |F|^{k'} C_{j-i-1}^{k'} \\ &= |F|(1+|F|)^{j-i-1}\end{aligned}$$

$\mathcal{N}$ est un majorant du nombre de valeurs possibles pour un coefficient d'un produit quelconque de matrices $\tilde{C}_a\tilde{C}_b\ldots$. On a alors un majorant $\mathcal{M}$ du nombre d'éléments du semi-groupe $\mathcal{S} = \langle \tilde{C}_a, a \in \Sigma \rangle$ :

$$\mathcal{M} = \prod_{i<j\leq n} |F|(1+|F|)^{j-i-1}$$

où $n$ est la dimensions des matrices du semi-groupe. Les produits ne dépendant que de $j-i$, on regroupe les termes en fonction de cette différence :

$$\begin{aligned}\mathcal{M} &= \prod_{k=1}^{n-1} \Big(|F|(1+|F|)^{k-1}\Big)^{n-k} \\ &= \prod_{k=0}^{n-2} \Big(|F|(1+|F|)^{k}\Big)^{n-k-1} \\ &= |F|^{\sum_{k=0}^{n-2}(n-k-1)} (1+|F|)^{\sum_{k=0}^{n-2}(n-k-1).k}\end{aligned}$$

En calculant les deux exposants, l'expression devient :

$$\mathcal{M} = |F|^{\frac{n(n-1)}{2}} (1+|F|)^{\frac{n(n-1)(n-2)}{6}}$$

## 3.4 Simplification des matrices générant le semi-groupe par conjugaison diagonale

Le semi-groupe $\langle C_a, a \in \Sigma \rangle$ engendré par les matrices $C_a, a \in \Sigma$, est infini. Nous cherchons maintenant un changement de base qui permettrait de simplifier l'expression des éléments de la matrice.

Rappelons que nous appelons $C_n$ la matrice $C_{A[n]}$, c'est-à-dire la matrice associée à la $n^{ième}$ lettre de la séquence $A$. Tout d'abord nous devons multiplier la matrice $C_n$ par $\delta \otimes I$, pour obtenir des 0 sur la diagonale. On obtient la matrice $\tilde{C}_n$ :

$$(\tilde{C}_n)_{i,j} = \begin{cases} si\ i<j & \sigma(i,n) \otimes \delta^{-(|i-j|-2)} = \sigma(i,n) - (|i-j|-2).\delta \\ si\ i>j & \varepsilon = -\infty \\ si\ i=j & \delta^0 = 0 \end{cases}$$

Cherchons maintenant le changement de base qui permettrait de simplifier l'expression des éléments de la matrice. Posons :

$$D = \begin{pmatrix} e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \dots \\ \varepsilon & \delta^{-1} & \varepsilon & \varepsilon & \varepsilon & \dots \\ \varepsilon & \varepsilon & \delta^{-2} & \varepsilon & \varepsilon & \dots \\ \varepsilon & \varepsilon & \varepsilon & \delta^{-3} & \varepsilon & \dots \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \delta^{-4} & \dots \\ . & . & . & . & . & . \end{pmatrix}$$

On a alors :

$$D^{-1} = \begin{pmatrix} e & \varepsilon & \varepsilon & \varepsilon & \varepsilon & \dots \\ \varepsilon & \delta^{1} & \varepsilon & \varepsilon & \varepsilon & \dots \\ \varepsilon & \varepsilon & \delta^{2} & \varepsilon & \varepsilon & \dots \\ \varepsilon & \varepsilon & \varepsilon & \delta^{3} & \varepsilon & \dots \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \delta^{4} & \dots \\ . & . & . & . & . & . \end{pmatrix}$$

La multiplication à droite par $D^{-1}$ correspond à la multiplication de chaque colonne $j$ par $\delta^{j-1}$. De la même manière la multiplication à gauche par $D$ correspond à la multiplication de chaque ligne $i$ par $\delta^{-(i-1)}$. Le passage de $\tilde{C}_n$ à $D\tilde{C}_nD^{-1}$ s'effectue par la multiplication de chaque élément $(\tilde{C}_n)_{i,j}$ par $\delta^{j-i}$. La matrice $D\tilde{C}_nD^{-1}$ s'écrit :

$$C' = D\tilde{C}D^{-1} = \begin{pmatrix} e & & & & & \\ \varepsilon & e & & C'_{i,j} & & \\ \varepsilon & \varepsilon & e & & & \\ . & . & . & e & & \\ . & . & & \varepsilon & e & \\ \varepsilon & \varepsilon & \dots & \varepsilon & \varepsilon & e \end{pmatrix}$$

avec :

$$C'_{i,j} = \begin{cases} si\ i < j & \sigma(i,n) \otimes \delta^2 = \sigma(i,n) + 2\delta \\ si\ i > j & \varepsilon = -\infty \\ si\ i = j & \delta^0 = 0 \end{cases}$$

La taille du semi-groupe $\langle C'a, a \in \Sigma \rangle$ engendré par les matrices $C'_a, a \in \Sigma$, est évidemment la même que celle du semi-groupe $\langle \tilde{C}_a, a \in \Sigma \rangle$.

## 3.5 Automate de Cayley

Soit $\Sigma$ un alphabet de 4 ou 20 lettres. On associe à chaque lettre de l'alphabet une matrice $C_a, a \in \Sigma$. Chaque matrice dépend de la séquence requête $B$ fixée, de longueur $l_B$. Toutes ces matrices sont de taille $(l_B+1) \times (l_B+1)$. Le semi-groupe $\langle C_a, a \in \Sigma \rangle$ est infini (les termes diagonaux tendent vers $-\infty$). Après une transformation sur les matrices génératrices, on peut se placer dans le cas fini. Le semi-groupe $\langle \tilde{C}_a = \delta C_a, a \in \Sigma \rangle$ est fini.

On cherche à construire un automate qui associe à tout mot $w$ la matrice du semi-groupe qui correspond aux produits successifs des matrices associées aux lettres du mot $w$. Si $w = abc$, la

matrice associée à $w$ est donnée par $C = \tilde{C}_a \tilde{C}_b \tilde{C}_c$. Rappelons que le morphisme défini par l'équation 3.7, p. 95, permet de passer canoniquement des mots de $\Sigma^*$ vers le semi-groupe $\langle \tilde{C}_a, a \in \Sigma \rangle$. Pour deux mots $m_1$ et $m_2$ de $\Sigma^+$

$$\phi(m_1.m_2) = \phi(m_1) \otimes \phi(m_2).$$

Si on connaît la structure du semi-groupe des matrices, on peut construire l'automate suivant, dit de Cayley :

- le seul état initial, noté $e$, est l'état correspondant à la matrice identité,
- tous les états sont terminaux,
- il existe une transition de $\tilde{C}_1$ vers $\tilde{C}_2$ par la lettre «a» si $\tilde{C}_1 \otimes \tilde{C}_a = \tilde{C}_2$, où $\tilde{C}_1$ et $\tilde{C}_2$ sont deux matrices quelconques du semi-groupe.

Cet automate permet de calculer le score Needleman & Wunsch $N(A, B)$ pour toute séquence $A$. Puisque l'automate de Cayley est fini, le score $N(A, B)$ entre deux séquences $A$ et $B$, de longueur $l_A$ et $l_B$ vaut

$$N(A, B) = \delta^{-l_A} \otimes X_0 \otimes \phi(A) \otimes \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon \\ e \end{pmatrix}.$$

$\phi(A)$ se lit directement sur l'automate de Cayley en **temps linéaire**. Une boucle linéaire permet de calculer le score dans ce cas très particulier où le semi-groupe est fini, à une normalisation des générateurs près par une même constante ($\delta^{-1}$).

Nous allons voir dans la section suivante, que cette propriété se généralise dans une certaine mesure : il suffit que l'orbite du vecteur initial $X_0 \otimes \phi(A)$, $A \in \Sigma^*$, ne contienne qu'un nombre fini de vecteurs non proportionnels.

## 3.6 Automate réduit

Plaçons-nous dans le cas d'un semi-groupe projectivement fini : $\langle C_a, a \in \Sigma \rangle$. Nous venons de voir que ce qui nous intéresse est l'orbite du vecteur initial. Rappelons que la dimension des vecteurs de l'orbite du vecteur initial est $l_B + 1$.

Construisons les classes d'équivalence des vecteurs de l'orbite du vecteur initial, par rapport à la relation de proportionnalité : deux éléments $a$ et $b$ de l'orbite appartiennent à la même classe d'équivalence si et seulement si il existe $\lambda \in \mathbb{R}_{max} \setminus \{-\infty\}$ tel que $a = \lambda \otimes b$. Comme le semi-groupe $\langle C_a, a \in \Sigma \rangle$ est projectivement fini, l'orbite du vecteur initial le sera. L'ensemble des classes d'équivalence est donc fini.

On peut alors construire un automate, dont les sommets correspondent à ces classes d'équivalence, et qui calcule le score $N(A, B)$ en un temps linéaire par rapport à la longueur de $A$.

- Chaque état $i$ de l'automate correspond à une classe d'équivalence de l'orbite du vecteur initial. On associe à chaque état un représentant de la classe d'équivalence. Le vecteur associé à l'état $i$ est nommé $u_i$.
- L'état initial est associé à la classe d'équivalence du vecteur initial $X_0 = (0, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$.

- Tous les états sont terminaux.
- Il existe une transition de l'état $i$ vers l'état $j$ par la lettre «$a$» si et seulement si il existe $\lambda \in \mathbb{R}_{max}$ tel que $u_j = \lambda \otimes (u_i \otimes C_a)$, autrement dit si les vecteurs $u_j$ et $u_i \otimes C_a$ sont propostionnels. Si $\lambda = e$, les deux vecteurs $u_i$ et $u_j$ sont égaux. Si $\lambda \neq e$, les deux vecteurs $u_i$ et $u_j$ sont proportionnels.
- A la transition de l'état $i$ vers l'état $j$ par la lettre «$a$», on associe le poids $\lambda$, qui représente le coefficient de proportionnalité entre le représentant de l'état $j$ et le vecteur $(u_i \otimes C_a)$ : $u_j = \lambda \otimes (u_i \otimes C_a)$. Le coefficient $\lambda$ peut être nul. C'est le cas quand $u_j$ et $(u_i \otimes C_a)$ sont égaux.

Cet automate permet de calculer le vecteur $X_0\phi(A)$ pour toute séquence $A \in \Sigma^*$. Cet automate est fini, puisque l'ensemble des classes d'équivalence l'est. Le vecteur $X_0\phi(A)$ pour une séquence $A$ de longueur $l_A$ vaut

$$X_0\phi(A) = \underbrace{X_0 C_{A_1} C_{A_2} \cdots C_{A_{l_A}}}_{X_{l_A}}$$

où $X_0 C_{A_1} C_{A_2} \cdots C_{A_{l_A}}$ se lit sur l'automate réduit en **temps linéaire**. En effet, l'automate donne directement la classe d'équivalence du vecteur $X_{l_A}$. Il ne reste plus qu'à calculer le coefficient de proportionnalité $\lambda$ entre le vecteur $X_{l_A}$ et le représentant de sa classe d'équivalence. Ce coefficient s'obtient en temps linéaire lors du parcours de l'automate :

$$\lambda = \bigotimes_{k=1}^{l_A} \lambda_k$$

où $\lambda_k$ est le coefficient de proportionnalité associé à la $k^{ième}$ transition du chemin dans l'automate.

**Remarque 3.6.1** *Ce type d'automate est à rapprocher des automates à multiplicité sur l'algèbre $(max, +)$. On se réfèrera à [141][96].*

## 3.7 Automate réduit associé à l'algorithme Needleman & Wunsch

On cherche à calculer pour tout mot $w \in \Sigma^*$, $X_0 \otimes \psi(w)$, où l'application $\psi$ est l'unique morphisme tel que $\forall a \in \Sigma, \psi(a) = C_a$. Puisque le semi-groupe $\langle C_a, a \in \Sigma \rangle$ est projectivement fini, l'orbite de $X_0$ par le semi-groupe, le sera. On cherche à construire alors l'automate réduit associé aux matrices $C_a, a \in \Sigma$.

Une fois qu'on connaît la structure de l'orbite du vecteur initial dans ce semi-groupe, la comparaison de la séquence $B$, avec n'importe quelle autre séquence $A$ revient à parcourir dans cette orbite un chemin correspondant à la séquence $A$. Autrement dit, on peut parcourir la séquence $A$, en gardant un pointeur sur un élément de l'orbite du vecteur initial correspondant et calculer rapidement le score NW.

### 3.7.1 Construction de l'automate réduit

On parcourt les mots de $\Sigma^*$, jusqu'à ce que l'automate réduit soit totalement construit. On a déjà montré que l'automate est fini, donc l'algorithme se termine. L'ordre choisi pour le parcours de $\Sigma^*$ est l'ordre alphabétique par taille, dit *ordre militaire* : le premier critère est la longueur du

mot, et le second est l'ordre alphabétique. Sur l'alphabet $\Sigma = \{a, b\}$, l'ordre militaire est : $e$, $a$, $b$, $aa$, $ab$, $ba$, $bb$, $aaa$, $aab$ ... où $e$ est le mot vide.

**Définition 3.7.1** *Soit un alphabet $\Sigma = \{a_1, a_2, \ldots, a_l\}$, et $w$ un mot quelconque de $\Sigma^*$. Tout mot de la forme $wa_i, a_i \in \Sigma$, sera appelé* fils *de $w$.*

A chaque étape on maintient deux listes d'états : la liste $L$ des états construits, et celle des états dont on n'a pas encore explorer les fils, notée $E$.

**Algorithme de construction de l'automate réduit**

1. Construction des matrices $C_a, a \in \Sigma$.
2. L'état initial $e$ est associé au vecteur initial $X_0 = (0, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$. Le représentant choisi est $X_0$. Initialisation des deux listes :
$$L = \{e\}$$
$$E = \{e\}$$
3. Pour chaque état $i$ (dont le vecteur représentant est $u_i$) de $E$
   - Pour chaque lettre $a$ de l'alphabet, on explore les fils de l'état $i$ :
     - Calcul de $u = u_i \otimes C_a$.
     - S'il existe $\lambda \in \mathbb{R}_{max}$ et $j \in L$ tel que $u_j = \lambda \otimes u$ c'est-à-dire si $u$ est proportionnel à l'un des vecteurs $j$ de $L$, alors on crée une transition de $i$ vers $j$, de label $a$ et de coefficient $\lambda$.
     - Sinon on crée un nouvel état $k$, que l'on rajoute dans $E$. On choisit comme vecteur représentant de ce nouvel état $k$, le vecteur $u_k = u$.
   - On enlève $i$ de la liste $E$.
4. Terminaison lorsque la liste $E$ est vide.

Cet algorithme se termine puisque l'automate réduit est fini.

Lors de la phase 3 de l'algorithme, on doit effectuer la multiplication $u = u_i \otimes C_a$. Ce calcul direct est très coûteux, sa complexité est $O((l_B + 1) \times (l_B + 1))$. Il vaut mieux revenir à l'équation en $Y$ : $Y = YB \oplus u_i A_a$ (Cf. éq. 3.4). La solution de cette équation est donnée par les relations de la programmation dynamique, qui donnent le résultat en $O(l_B + 1)$.

Cet automate reconnaît tous les mots de $\Sigma^*$. Il est déterministe. La figure 3.1 représente l'automate dans un cas simple.

**Remarque 3.7.1** *L'algorithme implémenté est l'un des algorithmes les plus naïfs, puisqu'il parcourt les mots du monoïde par ordre militaire. Il existe des algorithmes de calcul de semi-groupes finis plus efficaces [43].*

La table ci-dessous donne quelques exemples de la taille du semi-groupe et de l'automate réduit pour différentes séquences requêtes. De plus la table donne la **profondeur** associée à l'automate réduit, qui est la longueur maximale des mots qu'on a dû parcourir (par ordre militaire), pour construire l'automate. Puisqu'on utilise l'ordre militaire, aucun autre ordre ne permettra de construire l'automate en utilisant que des mots de longueurs strictement inférieures à la profondeur.

On s'est restreint au problème appliqué à l'ADN, l'alphabet est de taille 4. La matrice de similarité est $+10$ sur la diagonale, et $-9$ ailleurs, et $gapo = gape = -10$. La taille de l'automate réduit est nettement plus petite que celle de l'automate de Cayley.

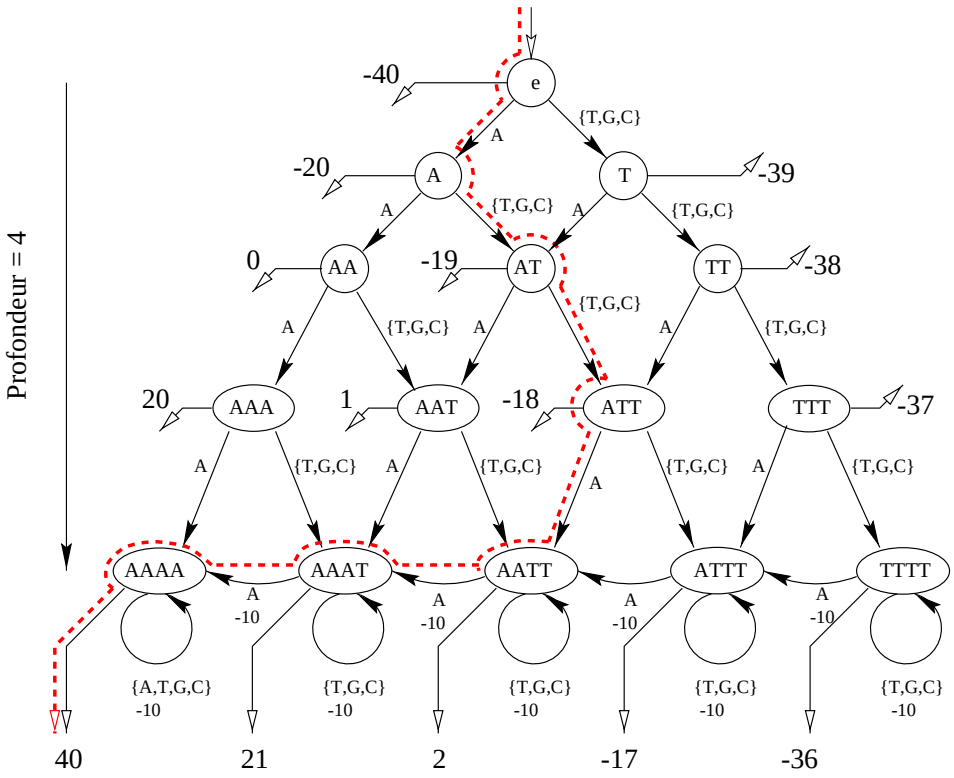



FIG. 3.1 - **Un automate dans le semi-groupe engendré par les matrices** $(\max, +)$. *La comparaison d'une séquence contre une banque de séquence par l'algorithme Needleman & Wunsch revient à parcourir un automate à l'aide de chaque séquence de la banque. La séquence* requête *choisie ici est* **AAAA**, *et les pénalités sont respectivement :* $\sigma(a, a) = 10 \quad \forall a \in \{A, T, G, C\}$, $\sigma(a, b) = -9 \quad \forall a, b \in \{A, T, G, C\}$ *pour* $a \neq b$ *et* $gapo = gape = -10$.

**Parcours de l'automate :** *L'état initial est e. Pour toute lettre «a» lue dans la séquence cible, on se déplace dans l'automate à l'aide de la transition de label «a». A chaque étape, on maintient le produit, au sens* $(max, +)$, *des coefficients de proportionnalité (poids associés aux transitions). Les coefficients non marqués valent tous* 0.

*Tous les états sont finaux. Le score de chacun d'eux, marqué par une flèche sortante, correspond à la dernière composante du vecteur associé.*

**Le score final** *est obtenu en multipliant, au sens* $(max, +)$, *le score associé à l'état final par le produit, au sens* $(max, +)$, *des coefficients de proportionnalité. Par exemple, le chemin en pointillé correspond au parcours de l'automate pour la séquence* **ATGAAA**. *Le score final vaut 20 = 40 -10 -10.* **La structure** *de cet automate est liée à la séquence* requête *AAAA. Les transitions avec un coefficient de proportionnalité non nul, peuvent, dans le cas général, pointer sur un état de profondeur inférieure (exemple : AAAT à AT).*

| mots | taille du semi-groupe des matrices | taille de l'orbite du vecteur initial | profondeur |
|---|---|---|---|
| AAAA | 14 | 15 | 4 |
| ATTA | 83 | 43 | 5 |
| ATGC | 400 | 84 | 6 |
| ATCGA | 1570 | 199 | 8 |
| ATCGAT | 5328 | 439 | 9 |
| ATCGATC | 15271 | 919 | 10 |
| ATCGATCG | 39047 | 1873 | 12 |

TAB. 3.1 - **Taille des semi-groupes engendrés par les matrices** (max, +) *pour l'algorithme de Needleman & Wunsch. La taille de l'orbite représente la taille de l'automate réduit.*

### 3.7.2 Calcul du score NW :

On parcourt grâce à l'automate la séquence *cible* lettre à lettre. Soit $A$ la séquence *cible* de longueur $l_A$. Au fur et à mesure qu'on avance dans la séquence, on maintient le produit (au sens $(max, +)$) des poids des transitions successives : $p_l = \bigotimes_{k=1}^{l} \lambda_k$ où $\lambda_k$ est le coefficient de proportionnalité pour la $k^{ième}$ transition.

L'automate reconnaît la séquence *cible*. Supposons que l'état terminal associé soit $k$. Le vecteur $u_k$ donne à un coefficient près le score NW. La dernière ligne du tableau habituel de l'algoritme NW est obtenue en multipliant (au sens $(max, +)$) le vecteur $u_k$ par le produit des poids des transitions successives. Le score NW est donc égal au produit (au sens $(max, +)$) de la dernière coordonnée de $u_k$ par le produit des poids des transitions successives.

$$N(A,B) = u_k[l_B] \otimes \left(\bigotimes_{k=1}^{l_A} \lambda_k\right) = X_0 \otimes \psi(A) \otimes \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon \\ e \end{pmatrix}$$

On a alors le résultat suivant :

**Théorème 3.7.1** *Soit $B$ une séquence. Une fois l'automate associé à $B$ construit, le calcul du score Needleman & Wunsch de $B$ contre une séquence quelconque $A$ est* linéaire *en fonction de la longueur de la séquence $A$.*

Rappelons que l'automate dépend de la séquence $B$, et que tous les vecteurs de l'orbite du vecteur initial dans le semi-groupe, sont de taille $l_B + 1$, dont les coordonnées sont numérotées de 0 à $l_B$.

## 3.8 Recherche de la meilleure occurrence d'un mot au sens NW

On compare une petite séquence (mot) à une grande séquence. La question est maintenant de déterminer les positions dans la grande séquence, qui correspondent aux sous-séquences les plus proches du mot. Soit $B$ le mot de longueur $l_B$, et $A$ de longueur $l_A$, la séquence dans laquelle on

cherche la sous-séquence qui ressemble le plus au mot $B$. La récurrence est la même que pour NW (Cf. éq. 3.1), mais l'algorithme en diffère par les initialisations (à comparer avec l'équation 3.2) :

$$\begin{array}{lcll} N(0,0) & = & 0 & \\ N(i,0) & = & 0 & \forall i \in [1, l_A] \\ N(0,j) & = & g(j) = -\delta \times j & \forall j \in [1, l_B] \end{array} \tag{3.8}$$

Notons que le score qui nous intéresse n'est pas la dernière case du tableau de la programmation dynamique. On s'intéresse au maximum de la dernière colonne. Cela signifie que les coûts dus aux insertions dans la séquence $A$ après l'occurrence de $B$ sont nuls.

En reprenant les notations de la section précédente, on a la relation suivante entre les vecteurs $X_n$ et $X_{n-1}$ :

$$X_n = X_{n-1} \otimes A_n \quad \oplus \quad X_n \otimes B_n$$

avec

$$B_n = \begin{pmatrix} . & \delta^{-1} & . & . & . \\ . & . & \delta^{-1} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \delta^{-1} \\ . & . & . & . & . \end{pmatrix} \qquad A_n = \left(\begin{array}{c|cccc|c} e & \sigma_{1,n} & . & \dots & . & . \\ \hline . & \delta^{-1} & \sigma_{2,n} & \dots & . & . \\ . & . & \delta^{-1} & \ddots & . & . \\ . & . & . & \ddots & \sigma_{l_B-1,n} & . \\ . & . & . & \ddots & \delta^{-1} & \sigma_{l_B,n} \\ \hline . & . & . & . & . & e \end{array}\right)$$

Il est à remarquer que la matrice $A_n$ n'est pas la même que celle de la récurrence 3.4. Le premier et le dernier élément de la diagonale de la matrice sont nuls. Le premier permet d'imposer $X_n(0) = 0, \forall n \in [1, l_A]$, tandis que le dernier permet de ne pas pénaliser les insertions/délétions en fin de séquence. Le score de la meilleure occurrence au sens NW est lu dans la dernière coordonnée du dernier vecteur. En posant $D_n = A_n \otimes B^*$, on a la récurrence suivante :

$$X_n = X_{n-1} D_n \tag{3.9}$$

L'initialisation de la récurrence est $X_0 = (0, \delta^{-1}, \delta^{-2}, \dots, \delta^{-l_B})$. La matrice $D_n$ est de la forme :

$$D_n = \left(\begin{array}{c|ccc|c} e & & D_{1,j} & & D_{1,l_B+1} \\ \hline \varepsilon & \delta^{-1} & & & \\ \varepsilon & \varepsilon & \delta^{-1} & & D_{i,l_B+1} \\ . & . & . & \delta^{-1} & \\ . & . & & \varepsilon \quad \delta^{-1} & \\ \hline \varepsilon & \varepsilon & \dots & \varepsilon \quad \varepsilon & e \end{array}\right) \qquad \begin{array}{l} D_{i,j} \\ \\ \\ \\ \\ \\ \end{array}$$

avec :

$$D_{i,j} = \begin{cases} si\ i = j = 1 & e & = & 0 \\ si\ i = j = l_B + 1 & e & = & 0 \\ si\ i = 1 \neq j & \left(\delta^{-1} \oplus \sigma(i,n)\right) \otimes \delta^{-(|i-j|-1)} & = & \max\left(-\delta, \sigma(i,n)\right) - (|i-j|-1).\delta \\ si\ i < j & \left(\delta^{-2} \oplus \sigma(i,n)\right) \otimes \delta^{-(|i-j|-1)} & = & \max\left(-2\delta, \sigma(i,n)\right) - (|i-j|-1).\delta \\ si\ i > j & \varepsilon & = & -\infty \\ si\ i = j \begin{pmatrix} i \neq 1, \\ i \neq l_B+1 \end{pmatrix} & \delta^{-1} & = & -\delta \end{cases}$$

Chaque matrice $D_a, a \in \Sigma$, diffère de la matrice correspondante du problème NW. Cette différence a beaucoup d'incidences sur le semi-groupe. La plus importante est certainement que l'orbite du vecteur initial dans le semi-groupe $\langle D_a, a \in \Sigma \rangle$ est fini, mais le semi-groupe reste infini.

**Théorème 3.8.1** *Soit* $X_0 \in (\mathbb{R}_{max} \setminus \{-\infty\})^{l_B+1}$ *à coordonnées toutes finies :* $X_0(i) \neq \varepsilon \ \forall i \in [1, l_B + 1]$. *L'ensemble* $\{X_0 D_{a_1} D_{a_2} \cdots D_{a_k}, a_1 a_2 \cdots a_k \in \Sigma^*\}$ *est fini.*

**Démonstration :**

Construisons d'abord l'automate associé aux graphes des matrices de précédence.

- L'ensemble des états de l'automate est l'ensemble des coordonnées d'un vecteur ligne $\mathcal{S} = \{0, 1, 2, \ldots, l_B\}$.
- Il y a autant de transitions de $i$ vers $j$ qu'il existe de lettres $a \in \Sigma$ telles que le graphe de précédence associé à la matrice $D_a$ contient l'arc $(i \longrightarrow j)$. Les transitions seront notées $(i \xrightarrow{a} j)$ où $a$ désigne une lettre telle que $(D_a)_{i,j} \neq \varepsilon$.
- De plus on associe un poids à chacune de ces transitions. La transition $(i \xrightarrow{a} j)$ aura un poids égal à $(D_a)_{i,j}$.

1. Montrons que l'ensemble des coefficients $(1, j)$ d'un produit de matrices $\{(D_{a_1} D_{a_2} \cdots D_{a_k})_{1j}, a_1 a_2 \cdots a_k \in \Sigma^*\}$ est fini.
   - $(D_{a_1} D_{a_2} \cdots D_{a_k})_{1j}$ est le poids maximum pour un chemin de label $a_1 a_2 \cdots a_k$ allant de 1 à $j$.
   - Soit $V_j$ l'ensemble des valeurs possibles pour le poids d'un chemin élémentaire quelconque allant de 1 à $j$. Le nombre de chemins élémentaires étant fini, l'ensemble $V_j$ est fini.
     Soit $M_j$ et $m_j$ le maximum et le minimum de $V_j$.
   - Soit $V'_j$ l'ensemble des valeurs possibles pour le poids d'un chemin quelconque allant de 1 à $j$. Les boucles sont maintenant permises. Un chemin avec boucles se décompose en un chemin sans boucle et un certain nombre de boucles.
     Dans le graphe, toutes les boucles (sauf celles allant de 1 à 1) ont le même poids : $\delta^{-1}$. La première boucle ayant un poids nul, on peut toujours se ramener à un chemin ne contenant pas par cette première boucle.
     Le poids d'un chemin avec boucle sera égal à un poids d'un chemin sans boucle, plus une constante, multiple de $\delta^{-1}$.

     $$V'_j \subset V_j \cup \delta^{-1} \otimes V_j \cup \delta^{-2} \otimes V_j \cup \cdots$$

     où $\delta^{-k} \otimes V_j$ est l'ensemble des valeurs de $V_j$ multipliées par $\delta^{-k}$.
     $V'_j$ est majoré par $M_j$, le maximum de $V_j$.
   - Prenons un chemin quelconque de 1 à $j$ avec $k$ boucles de poids $\delta^{-1}$. Notons $c$ le poids de ce chemin. Soit $k_0^j = \lfloor \frac{M_j - m_j}{\delta} \rfloor + 1$. Dès que ce chemin passe par plus de $k_0^j$ boucles de poids $\delta^{-1}$, le poids de ce chemin est inférieur à $m_j$. En effet, $c = c_e - k \times \delta$, avec $c_e \in V_j$. $c < M_j - (M_j - m_j) = m_j$.
     Pour le même label, il existe un chemin de meilleur poids : celui où toutes les boucles sont au début, et où il n'y a plus de boucles de poids $\delta^{-1}$. En effet, ce chemin à un poids appartenant à $V_j$, donc minoré par $m_j$.

- On peut se restreindre aux chemins ayant moins de $k_0^j$ boucles de poids $\delta^{-1}$.
  Il y a un nombre fini de chemins ayant au plus $k_0^j$ boucles de poids $\delta^{-1}$, si on ne compte pas les chemins ayant des boucles de poids nul. Les boucles de poids nul n'influent pas sur le poids du chemin.
  Donc l'ensemble des poids des chemins optimaux allant de 1 à $j$, pour un mot quelconque $a_1 a_2 \cdots a_k$ est inclue dans

$$V_j{}' \subset V_j{}'' = V_j \cup \delta^{-1} \otimes V_j \cup \delta^{-2} \otimes V_j \cup \cdots \cup \delta^{-k_0^j} \otimes V_j$$

  $V_j{}''$ est fini, donc $V_j{}'$ l'est aussi.

2. Montrons maintenant que $\{(X_0 D_{a_1} D_{a_2} \cdots D_{a_k})_l, a_1 a_2 \cdots a_k \in \Sigma^*\}$ est fini. Posons $w = a_1 a_2 \cdots a_k$.
   - On cherche à calculer $X_0 \otimes \phi(w)$ où l'application $\phi$ est l'unique morphisme tel que $\forall a \in \Sigma, \phi(a) = D_a$.

$$(X_0 \otimes \phi(w))_l = \bigoplus_{i=1}^{l_B+1} X_0(i) \otimes (\phi(w))_{il}$$

     correspond au maximum des poids des chemins, de label $w$, allant de $i$ à $l$, où un chemin commençant en $i$ a une pénalité de $X_0(i)$.
   - $U = \{(X_0 \otimes \phi(w))_l, w \in \Sigma^*\}$ est majoré.
     Un majorant de $(X_0 \otimes \phi(w))_l$ est $\max_{i=1}^{l_B+1}(X_0(i)) + K$, où $K$ est le maximum des poids des chemins élémentaires.
   - $U = \{(X_0 \otimes \phi(w))_l, w \in \Sigma^*\}$ est minoré.

$$(X_0 \otimes \phi(w))_l \geq X_0(1) \otimes (\phi(w))_{1l} \geq X_0(1) \otimes m_l$$

   - $U = \{(X_0 \otimes \phi(w))_l, w \in \Sigma^*\}$ est fini.
     $(X_0 \otimes \phi(w))_l$ se décompose en un poids d'un chemin élémentaire, un certain nombre de boucles et une pénalisation.

$$\begin{aligned}(X_0 \otimes \phi(w))_l &= X_0(i) \otimes \phi(w)_{il} \\ &= X_0(i) \otimes \delta^{-k} \otimes c_e\end{aligned}$$

     où $c_e$ est le poids du chemin élémentaire. Comme $U$ est borné, $k$ est majoré. $c_e$ et $X_0(i)$ appartiennent à des ensembles finis, $k$ est borné, donc $U$ est fini.

Puisque $\{(X_0 D_{a_1} D_{a_2} \cdots D_{a_k})_l,\ a_1 a_2 \cdots a_k \in \Sigma^*\}$ est fini, $\{X_0 D_{a_1} D_{a_2} \cdots D_{a_k}, a_1 a_2 \cdots a_k \in \Sigma^*\}$ est aussi fini.□

**Corollaire 3.8.1** *L'orbite du vecteur initial* $X_0 = (0, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$ *dans le semi-groupe* $\langle D_a, a \in \Sigma \rangle$ *est fini.*

**Corollaire 3.8.2** *L'automate réduit est fini.*

On remarque que l'orbite du vecteur initial est beaucoup plus importante dans le cas de la recherche de la meilleure occurrence d'un mot au sens NW, que dans le cas de l'algorithme NW (comparer les tableaux 3.2 et 3.1). Il ne s'agit pour l'instant que d'une constatation qu'il faudrait quantifier. Cela provient entre autre du fait que, dans ce nouveau problème, la relation d'équivalence

| mots | taille de l'orbite du vecteur initial | profondeur |
|---|---|---|
| AAAA | 164 | 14 |
| ATTA | 305 | 12 |
| ATGC | 365 | 10 |
| ATCGA | 1680 | 13 |
| ATCGAT | 6162 | 16 |
| ATCGATC | 23116 | 18 |
| ATCGATCG | 79205 | 23 |

TAB. 3.2 - **Taille des automates réduits engendrés par les matrices** $(\max, +)$ *dans le cas de la recherche de la meilleure occurrence d'un mot au sens Needleman & Wunsch.*

$\sim$ introduite dans le paragraphe 3.3, n'est plus utile ici. Aucun élément de l'orbite du vecteur initial $X_0 = (0, \delta^{-1}, \delta^{-2}, \ldots, \delta^{-l_B})$, ne peut être proportionnel (au sens $(max, +)$) à un autre vecteur de cette même orbite. En effet, chaque vecteur de l'orbite a pour première coordonnée «0», et si deux vecteurs sont proportionnels, ils sont égaux.

La remarque précédente s'interprète aussi par le fait que, contrairement au cas NW, dans le cas de la recherche d'une occurrence d'un mot, on ne tient pas compte de la position de cette occurrence dans la séquence $A$.

**Corollaire 3.8.3** *Soit $B \in \Sigma^*$. Le score de la meilleure occurrence du mot $B$ au sens NW dans une séquence quelconque, ne prend qu'un nombre fini de valeurs.*

> **Démonstration** : Dans l'automate réduit, toutes les transitions ont un poids nul. Donc le coefficient de proportionnalité $\left(\otimes_{k=1}^{l_A} \lambda_k\right)$ est nul. Il n'y a donc pas plus de scores que d'états. □

On peut diminuer la taille de l'automate. Lors de la construction des matrices $A_n$, on a introduit deux 0, en position $(1,1)$ et $(l_B + 1, l_B + 1)$, correspondant respectivement aux non-pénalisations des insertions/délétions de début et fin de séquences *cibles*. Le dernier élément peut être gardé à $\delta^{-1}$, comme dans le cas de NW. Les pénalisations en fin de séquences ne sont plus nulles.

Si on construit l'automate associé à ces nouvelles matrices, cet automate possède moins d'états (comparer les tableaux 3.2 et 3.3). Le score obtenu à la fin du parcours de la séquence *cible* ne sera pas le score souhaité. On peut obtenir le score désiré en maintenant une variable supplémentaire lors du parcours de l'automate. A chaque étape, on garde le maximum des scores obtenus jusque-là. Lorsqu'on a parcouru toute la séquence, ce maximum correspond au score de la meilleure occurrence de la séquence *requête*. Ce calcul est en $O(l_A)$, c'est-à-dire de même complexité que le parcours de l'automate.

## 3.9 L'algorithme de Smith & Waterman

La formulation de l'algorithme de Smith & Waterman doit être faite avec un peu de soin, parce que le score SW n'est pas forcément dans la dernière ligne du tableau. Si on ne s'intéresse qu'à transcrire l'algorithme de Smith & Waterman dans l'algèbre (max, +), en se plaçant dans le

| mots | taille de l'orbite du vecteur initial | profondeur |
|---|---|---|
| AAAA | 87 | 11 |
| ATTA | 139 | 9 |
| ATGC | 191 | 8 |
| ATCGA | 599 | 12 |
| ATCGAT | 1840 | 13 |
| ATCGATC | 5489 | 16 |
| ATCGATCG | 15842 | 19 |

TAB. 3.3 - **Taille des automates réduits engendrés par les matrices** $(\max,+)$ *dans le cas de la recherche de la meilleure occurrence d'un mot au sens Needleman & Wunsch. La taille de l'orbite du vecteur initial a diminué, parce que les matrices génératrices ont été modifiées : l'élément* $(l_B+1, l_B+1)$ *vaut* $\delta^{-1}$.

cas simple où la fonction de pénalisation des insertions/délétions est linéaire, on a la récurrence suivante :

$$\begin{array}{rcl} S_{i,j} & = & \max\Big(S_{i-1,j-1}+\sigma_{i,j} \quad , \quad S_{i-1,j}-\delta \quad , \quad S_{i,j-1}-\delta \quad , \quad 0\Big) \\ & = & \Big(S_{i-1,j-1}\otimes\sigma_{i,j}\Big) \quad \oplus \quad \Big(\delta^{-1}\otimes S_{i-1,j}\Big) \quad \oplus \quad \Big(\delta^{-1}\otimes S_{i,j-1}\Big) \quad \oplus \quad 0 \end{array} \tag{3.10}$$

avec les conditions initiales :

$$\left\{ \begin{array}{rcll} S(0,0) & = & 0 & \\ S(i,0) & = & 0 & \forall i \in [1, l_A] \\ S(0,j) & = & 0 & \forall j \in [1, l_B] \end{array} \right. \tag{3.11}$$

En posant $Y_n$ le vecteur ligne de taille $l_B+1$ correspondant à la $n^{ième}$ ligne du tableau habituel de l'algorithme SW, on a :

$$Y_n \quad = \quad Y_{n-1}\otimes A_n \quad \oplus \quad Y_n \otimes B_n \quad \oplus C \tag{3.12}$$

avec

$$C \quad = \quad (0\ ,\ 0\ ,\ \ldots\ ,\ 0)$$

$$B_n = \begin{pmatrix} . & \delta^{-1} & . & . & . \\ . & . & \delta^{-1} & . & . \\ . & . & . & \ddots & . \\ . & . & . & . & \delta^{-1} \\ . & . & . & . & . \end{pmatrix} \qquad A_n = \left(\begin{array}{c|cccc} e & \sigma_{1,n} & . & \ldots & . \\ \hline . & \delta^{-1} & \sigma_{2,n} & \ldots & . \\ . & . & \delta^{-1} & \ddots & . \\ . & . & . & \ddots & \sigma_{l_B,n} \\ . & . & . & . & \delta^{-1} \end{array}\right)$$

Le coefficient $(1,1)$ de la matrice $A_n$ vaut $e$ pour imposer dans la récurrence sur les vecteurs $Y_n$, $Y_n(0) = 0, \forall n \in [1, l_A]$.

**Remarque 3.9.1** *On peut changer la définition de* $A_n$ *en posant* $A_n(1,1) = \delta^{-1}$, *sans perdre la propriété* $Y_n(0) = 0, \forall n \in [1, l_A]$. *Dans ce cas-là, le* 0 *de la première coordonnée sera imposé par la comparaison au vecteur* $C$, *dont la première coordonnée vaut* 0.

$B_n$ est indépendant de $n$, et $B^*$ est défini de la même manière (Cf. éq. 3.5). On a $C \otimes B^* = C$. En posant $D_n = A_n \otimes B^*$, on a la récurrence suivante :

$$\begin{aligned} Y_n &= Y_{n-1} \otimes D_n \ \oplus \ C \otimes B^* = Y_{n-1} \otimes D_n \ \oplus \ C && (3.13) \\ Y_0 &= (\,0\,,\,0\,,\,\ldots\,,\,0) && (3.14) \end{aligned}$$

Il s'agit d'un *système dynamique affine*. Les premières itérations sont les suivantes :

$$\begin{aligned} Y_1 &= Y_0 D_1 \oplus C \\ Y_2 &= Y_1 D_2 \oplus C = Y_0 D_1 D_2 \quad \oplus C D_2 \quad \oplus C \\ Y_3 &= Y_2 D_3 \oplus C = Y_0 D_1 D_2 D_3 \oplus C D_2 D_3 \oplus C D_3 \oplus C \\ Y_n &= \bigoplus_{k=1}^{n+1} C \bigotimes_{i=k}^{n} D_i && (3.15) \end{aligned}$$

avec la convention :

$$\bigotimes_{i=n+1}^{n} D_i = I$$

Les vecteurs $C \overset{n}{\underset{i=k}{\otimes}} D_i$ sont associés aux suffixes de la sous-chaîne $A[1,n]$. Le calcul précédent correspond à chercher parmi tous les suffixes du mot considéré ($A[1,n]$) celui pour lequel le score est maximal. Ce calcul doit être effectué pour tous les préfixes $A[1,k]$, $\forall k \in [1, l_A]$. Dans le cas de l'algorithme Smith & Waterman, il faut regarder le score maximal dans tout le tableau. Cette opération peut se formaliser par :

$$SW = \bigoplus_{k=0}^{l_A} |Y_k| \tag{3.16}$$

où

$$|Y_k| = \bigoplus_{i=1}^{l_B+1} Y_k(i) \tag{3.17}$$

On prend le maximum de la norme des vecteurs $Y_k$ (éq. 3.16), où la norme est définie par le maximum des coordonnées du vecteur (éq. 3.17).

### Reformulation :

On peut passer à une écriture linéaire en augmentant de 1 la dimension du système. En effet, tous les termes des vecteurs $(Y_l)_{l=1\ldots l_A}$ sont comparés à 0. En rajoutant un élément qui vaut 0, on peut écrire :

$$(e, Y_n) = (e, Y_{n-1}) \otimes \left(\begin{array}{c|c} e & (e,e,e,\ldots) \\ \hline \varepsilon & \\ \varepsilon & \\ \varepsilon & \quad D_n \\ \varepsilon & \\ \vdots & \end{array}\right) \tag{3.18}$$

On pose $(\tilde{D}_\alpha)_{\alpha\in\Sigma}$ l'ensemble des matrices de dimensions $l_B+2$ obtenues en augmentant ainsi la dimension de 1.

**Théorème 3.9.1** *L'orbite du vecteur initial $X_0=(0,0,\ldots,0)$ dans le semi-groupe $\langle \tilde{D}_\alpha, \alpha\in\Sigma\rangle$ est fini.*

**Démonstration** : Toutes les matrices génératrices sont triangulaires. Les éléments diagonaux sont tous négatifs, sauf le premier qui est nul. Les caractéristiques nécessaires à la démonstration du théorème 3.8.1, sont réunies, la même démarche donne le résultat. □

Dans cette formulation, on a calculé toutes les valeurs de la matrice des scores, mais on n'a pas gardé le meilleur score du tableau. On a besoin d'une variable supplémentaire servant de *mémoire* pour stocker à chaque étape le score réalisant le maximum des cellules du tableau déjà parcourues. A chaque étape, on stocke dans une mémoire, la valeur maximale trouvée jusque là. Posons $Y'_n=(e,Y_n,M_{n-1})$ où $M_{n-1}$ représente le meilleur score d'alignement local pour comparer la séquence B contre $A[1,n-1]$. On a les relations suivantes :

$$M_n = \max\left(e, M_{n-1}, \bigoplus_{i=1}^{l_B+1}\left(Y_n(i)\right)\right) \tag{3.19}$$

$$(e,Y_n,M_{n-1}) = (e,Y_{n-1},M_{n-2}) \left(\begin{array}{c|c|c} e & (e,e,e,\ldots) & e \\ \hline \varepsilon & & e \\ \varepsilon & & e \\ \varepsilon & D_n & e \\ \varepsilon & & e \\ \vdots & & \vdots \\ \hline \varepsilon & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \end{array}\right) \tag{3.20}$$

Dans cette formulation, nous avons un décalage d'indices entre le vecteur $Y_n$ et la mémoire $M_{n-1}$. Posons $\tilde{Y}_n=(e,Y_n,M_n)$. On a la relation suivante :

$$\tilde{Y}_n = \tilde{Y}_{n-1} \underbrace{\left(\begin{array}{c|c|c} e & (e,e,e,\ldots) & \varepsilon \\ \hline \varepsilon & & \varepsilon \\ \varepsilon & & \varepsilon \\ \varepsilon & D_n & \varepsilon \\ \varepsilon & & \varepsilon \\ \vdots & & \vdots \\ \hline \varepsilon & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \end{array}\right)}_{E_n} \oplus \tilde{Y}_n \underbrace{\left(\begin{array}{c|c|c} e & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \\ \hline \varepsilon & & e \\ \varepsilon & & e \\ \varepsilon & I & e \\ \varepsilon & & e \\ \vdots & & \vdots \\ \hline \varepsilon & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \end{array}\right)}_{F} \tag{3.21}$$

$F$ est indépendant de $n$. Le graphe de précédence de $F$ ne contient aucun circuit de coût positif. La matrice $F^*$ existe et se calcule simplement. $F$ est idempotente, on a $F=F^2$, et $F^*=F$. La

solution de l'équation 3.21 est donnée par :

$$\begin{aligned}
\tilde{Y}_n &= \tilde{Y}_{n-1} \otimes \underbrace{\left(\begin{array}{c|c|c} e & (e,e,e,\ldots) & \varepsilon \\ \hline \varepsilon & & \varepsilon \\ \varepsilon & & \varepsilon \\ \varepsilon & D_n & \varepsilon \\ \varepsilon & & \varepsilon \\ \vdots & & \vdots \\ \hline \varepsilon & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \end{array}\right)}_{E_n} \otimes \underbrace{\left(\begin{array}{c|c|c} e & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \\ \hline \varepsilon & & e \\ \varepsilon & & e \\ \varepsilon & I & e \\ \varepsilon & & e \\ \vdots & & \vdots \\ \hline \varepsilon & (\varepsilon,\varepsilon,\varepsilon,\ldots) & e \end{array}\right)}_{F^*} \\
&= \tilde{Y}_{n-1} \otimes \underbrace{E_n \otimes F^*}_{G_n} \\
&= \tilde{Y}_{n-1} \otimes G_n \qquad (3.22)
\end{aligned}$$

Le problème SW s'exprime exactement comme le problème NW, et la formulation à l'aide de l'automate est la même. Seule la matrice diffère, la dimension augmente de deux. En conséquence l'automate est plus compliqué à construire, et il est beaucoup plus gros.

## 3.10 Pénalité affine des insertions/délétions

Supposons maintenant que la pénalité d'une insertion/délétion ne soit pas linéaire mais affine. On pénalise l'alignement passant par une insertion/délétion de longueur $k$ par $g(k) = gapo + gape \times (k-1) \quad \forall k \geq 1$. On a vu que pour l'algorithme, le passage du cas linéaire au cas affine, entraînait le doublement de l'espace nécessaire (Cf. section 1.2.1) : au lieu de garder pour chaque case $(i,j)$, une seule variable, on était obligé de stocker deux variables. On devrait retrouver ici ce phénomène.

### 3.10.1 Algorithme Needleman & Wunsch

Puisque l'algorithme de Needleman & Wunsch est plus simple, nous reprenons d'abord cet algorithme dans le cas où la pénalité des insertions/délétions est affine. Reprenons la relation de récurrence initiale :

$$M_{k,l} = \max \left( \begin{array}{c} M_{k-1,l-1} + \sigma(k,l), \\ -go + \max\limits_{1 \leq i < k} (M_{k-i,l} - (i-1).ge), \\ -go + \max\limits_{1 \leq i < l} (M_{k,l-i} - (i-1).ge) \end{array} \right) \qquad (3.23)$$

Posons $X_k$ et $Y_k$ les vecteurs lignes définis par :

$$X_k(l) = M_{k,l} \qquad (3.24)$$

$$Y_k(l) = \max_{1 \leq i < k} \Big( X_{k-i}(l) - (go + (i-1).ge) \Big) \qquad (3.25)$$

Le vecteur $X_k$ représente la ligne $k$ du tableau habituel de l'algorithme Needleman & Wunsch. $X_k(l)$ est la valeur maximale pour aligner les séquences $A[1,k]$ et $B[1,l]$. Les valeurs $Y_k(l)$ sont les valeurs maximales pour aligner $A[1,k]$ et $B[1,l]$ sachant que l'alignement se termine par une verticale, c'est-à-dire une insertion dans la séquence $A$. Cette récurrence n'est valable que dans le cas où $go \geq ge$, contrainte non restrictive.

$$
\begin{aligned}
Y_{k+1}(l) &= \max_{1\le i<k+1}\Big(X_{k+1-i}(l) - (go + (i-1).ge)\Big) \\
&= \max\Big(\max_{2\le i<k+1}\Big(X_{k+1-i}(k) - go - (i-1).ge\Big),\ \ X_k(l) - go\Big) \\
&= \max\Big(Y_k(l) - ge\ \ ,\ \ X_k(l) - go\Big) \qquad (3.26)\\
X_{k+1}(l) &= \max\begin{pmatrix} X_k(l-1) + \sigma(l,k+1), \\ Y_{k+1}(l), \\ \max_{1\le i\le l}(X_{k+1}(l-i) - go - (i-1)ge) \end{pmatrix} \qquad (3.27)
\end{aligned}
$$

On peut réécrire les formules de récurrence sur les vecteurs $X_k$ et $Y_k$ :

$$
\begin{array}{rcccccl}
X_k &=& X_{k-1}A_k &\oplus& X_kB &\oplus& Y_k \\
Y_k &=& Y_{k-1}C &\oplus& X_{k-1}D &&
\end{array} \qquad (3.28)
$$

avec

$$
A_k = \begin{pmatrix}
. & \sigma_{1,k} & . & . & . \\
. & . & \sigma_{2,k} & . & . \\
. & . & . & \ddots & . \\
. & . & . & . & \sigma_{l_B,k} \\
. & . & . & . & .
\end{pmatrix} \qquad (3.29)
$$

$$
B = \begin{pmatrix}
. & -go & -go-1.ge & -go-2.ge & \dots & -go-(l_B-1).ge \\
. & . & -go & -go-1.ge & \dots & -go-(l_B-2).ge \\
. & . & . & -go & \dots & -go-(l_B-3).ge \\
 & & & & \ddots & \vdots \\
. & . & . & . & . & -go \\
. & . & . & . & . & .
\end{pmatrix} \qquad (3.30)
$$

$$
C = -ge \otimes I \qquad (3.31)
$$

$$
D = -go \otimes I \qquad (3.32)
$$

où $I$ est la *matrice identité* pour l'algèbre $\mathbb{R}_{max}^{n\times n}$. Le graphe de précédence de la matrice $B$ ne contient pas de circuit de coût positif, donc $B^*$ existe et se calcule simplement. En utilisant une notation matricielle par blocs, on obtient la récurrence suivante, dont la dimension est deux fois plus grande que dans le cas linéaire :

$$
(X_l, Y_l) = (X_{l-1}, Y_{l-1}) \left(\begin{array}{c|c} (C_l + D)B^* & D \\ \hline CB^* & C \end{array}\right) \qquad (3.33)
$$

La même recurrence est valable pour la recherche de la meilleure occurrence d'un motif dans une séquence. Il n'y a que les initialisations qui changent.

### 3.10.2 Algorithme Smith & Waterman

Reprenons la relation de récurrence initiale:

$$M_{k,l} = \max \begin{pmatrix} M_{k-1,l-1} + \sigma(k,l), \\ -go + \max\limits_{1\leq i<k}(M_{k-i,l} - i.ge), \\ -go + \max\limits_{1\leq i<l}(M_{k,l-i} - i.ge), \\ 0 \end{pmatrix} \tag{3.34}$$

Posons $X_k$ et $Y_k$ les vecteurs lignes définis par:

$$X_k(l) = M_{k,l} \tag{3.35}$$

$$Y_k(l) = \max_{1\leq i<k}\Big(X_{k-i}(l) - (go + (i-1).ge)\Big) \tag{3.36}$$

Le vecteur $X_k$ représente la ligne $k$ du tableau habituel de l'algorithme Smith & Waterman. $X_k(l)$ est la valeur maximale pour aligner des suffixes de $A[1,k]$ et de $B[1,l]$. Les valeurs $Y_k(l)$ sont les valeurs maximales pour aligner des suffixes de $A[1,k]$ et de $B[1,l]$ sachant que l'alignement se termine par une verticale, c'est-à-dire une insertion dans la séquence $A$. Cette récurrence n'est valable que dans le cas où $go \geq ge$, contrainte non restrictive.
On a alors les équations suivantes:

$$\begin{aligned} Y_{k+1}(l) &= \max_{1\leq i<k+1}\Big(X_{k+1-i}(l) - (go + (i-1).ge)\Big) \\ &= \max\Big(\max_{2\leq i<k+1}\Big(X_{k+1-i}(k) - go - (i-1).ge\Big),\ \ X_k(l) - go\Big) \\ &= \max\Big(Y_k(l) - ge\ \ ,\ \ X_k(l) - go\Big) \end{aligned} \tag{3.37}$$

$$X_{k+1}(l) = \max \begin{pmatrix} X_k(l-1) + \sigma(l,k+1), \\ Y_{k+1}(l), \\ \max_{1\leq i\leq l}(X_{k+1}(l-i) - go - (i-1)ge), \\ 0 \end{pmatrix} \tag{3.38}$$

En rappelant que les matrices $A_k$, $B$, $C$ et $D$ sont celles définies par les équations 3.29, 3.30, 3.31 et 3.32, on peut réécrire les formules de récurrence sur les vecteurs $X_k$ et $Y_k$:

$$\begin{array}{rcl} X_k &=& X_{k-1}A_k \ \oplus\ X_kB \ \oplus\ Y_k \ \oplus\ (e,e,...,e) \\ Y_k &=& Y_{k-1}C \ \oplus\ X_{k-1}D \end{array} \tag{3.39}$$

En utilisant une notation matricielle par blocs, on obtient la récurrence suivante, dont la dimension est deux fois plus grande que dans le cas linéaire:

$$(e, X_l, Y_l) = (e, X_{l-1}, Y_{l-1}) \left(\begin{array}{c|c|c} e & (e,e,\ldots,e) & (e,e,\ldots,e) \\ \hline \varepsilon & & \\ \vdots & (C_l + D)B^* & D \\ \varepsilon & & \\ \hline \varepsilon & & \\ \vdots & CB^* & C \\ \varepsilon & & \end{array}\right) \tag{3.40}$$

Dans cette formulation, on a calculé toutes les valeurs de la matrice des scores, mais on n'a pas gardé le meilleur score du tableau. On a besoin d'une variable supplémentaire servant de *mémoire* pour stocker à chaque étape le score réalisant le maximum des cellules du tableau déjà parcourues. En utilisant la variable $M_n$ comme dans le cas où les pénalisations des insertions/délétions sont linéaires (Cf. section 3.9), on obtient la récurrence suivante :

$$(e, X_l, Y_l, M_{l-1}) \quad = \quad (e, X_{l-1}, Y_{l-1}, M_{l-2}) \begin{pmatrix} e & (e,e,\ldots,e) & (e,e,\ldots,e) & e \\ \hline \varepsilon & & & e \\ \vdots & (C_l + D)B^* & D & \vdots \\ \varepsilon & & & e \\ \hline \varepsilon & & & \varepsilon \\ \vdots & CB^* & C & \vdots \\ \varepsilon & & & \varepsilon \\ \hline \varepsilon & (\varepsilon,\varepsilon,\ldots,\varepsilon) & (\varepsilon,\varepsilon,\ldots,\varepsilon) & e \end{pmatrix} \tag{3.41}$$

Dans la dernière colonne de la matrice précédente, les $l_B + 1$ termes nuls ($\varepsilon$) proviennent du fait que le maximum ne peut pas être obtenu par un alignement se terminant par une insertion/délétion. Les $Y_k(l)$ correspondent à des scores de chemins se terminant par une insertion sur la séquence $A$.

Notons que l'on peut obtenir une équation portant sur les vecteurs $(e, X_l, Y_l, M_l)$, de telle manière qu'on n'ait pas de décalage d'indices entre celui de la *mémoire* et celui des scores.

$$(e, X_l, Y_l, M_l) \quad = \quad (e, X_{l-1}, Y_{l-1}, M_{l-1}) \begin{pmatrix} e & (e,e,\ldots,e) & (e,e,\ldots,e) & e \\ \hline \varepsilon & & & \varepsilon \\ \vdots & (C_l + D)B^* & D & \vdots \\ \varepsilon & & & \varepsilon \\ \hline \varepsilon & & & \varepsilon \\ \vdots & CB^* & C & \vdots \\ \varepsilon & & & \varepsilon \\ \hline \varepsilon & (\varepsilon,\varepsilon,\ldots,\varepsilon) & (\varepsilon,\varepsilon,\ldots,\varepsilon) & e \end{pmatrix} \tag{3.42}$$

$$\oplus \quad (e, X_l, Y_l, M_l) \begin{pmatrix} e & (\varepsilon,\varepsilon,\ldots,\varepsilon) & (\varepsilon,\varepsilon,\ldots,\varepsilon) & e \\ \hline \varepsilon & & & e \\ \vdots & I & \varepsilon & \vdots \\ \varepsilon & & & e \\ \hline \varepsilon & & & \varepsilon \\ \vdots & \varepsilon & I & \vdots \\ \varepsilon & & & \varepsilon \\ \hline \varepsilon & (\varepsilon,\varepsilon,\ldots,\varepsilon) & (\varepsilon,\varepsilon,\ldots,\varepsilon) & e \end{pmatrix} \tag{3.43}$$

Cette équation implicite se résout facilement comme dans le cas des pénalités linéaires (Cf. éq. 3.21 et 3.22, pp. 109 et 110). L'étoile de la deuxième matrice peut être calculée. On obtient alors une récurrence simple.

## 3.11 Mise en œuvre de l'algorithme avec automates

Il est clair que la taille de l'automate est la limitation de cet algorithme. Déjà dans les cas simples cités plus haut, où les pénalités des insertions/délétions sont linéaires et non affines, la taille de l'automate est importante (Cf. tableau 3.2).

Cependant, une application peut être mise en œuvre. Il s'agit en fait d'un cas totalement dégénéré de la programmation dynamique : la ***recherche d'occurrences d'un mot avec erreurs***. Plusieurs algorithmes permettent de trouver la ou les meilleures occurrences d'un mot. On peut citer cependant une alternative à la programmation dynamique, le programme ***agrep*** (**a**pproximative **grep** [137, 138, 139]).

Si on retient la programmation dynamique pour effectuer une recherche de la meilleure occurrence d'un mot avec $k$ erreurs, l'algorithme est équivalent à la programmation dynamique de Needleman & Wunsch, avec pour pénalités des insertions/délétions $gapo = gape = 1$, et pour matrice de similarité, la matrice ayant des 0 sur la diagonale et des $-1$ ailleurs. On cherche alors à maximiser le score NW, c'est-à-dire à minimiser le nombre d'erreurs dans l'alignement. Puisqu'on cherche la meilleure occurrence d'un mot au sens NW, les initialisations seront du type cité plus haut (éq. 3.8).

Le score sera négatif ou nul; il sera nul dans le cas de l'existence du mot recherché dans la séquence, et négatif dans le cas contraire. Dans ce dernier cas, la valeur du score sera le nombre d'erreurs, affecté du signe moins («$-$»).

L'algorithme se décompose en deux phases : construction de l'automate, puis parcours de la banque de données. La deuxième phase est très rapide, puisque lorsque l'automate est construit, l'algorithme de parcours de la banque de séquences est linéaire en temps par rapport à la taille de la banque. Pour chaque lettre d'une séquence $A$ de longueur $l_A$, on recherche l'état suivant de l'automate, et on compare le score courant avec le précédent. L'algorithme est en $O(l_A)$.

La construction de l'automate est plus complexe. La difficulté réside dans le fait que la taille du semi-groupe, plus exactement de l'orbite du vecteur initial par le semi-groupe des matrices, n'est pas connue, avant la construction de l'automate. On peut tout de même donner une complexité dépendant de la taille de l'automate (notée $N_{au}$). Pour chaque état de l'automate, on doit étudier toutes les transitions possibles à partir de l'état considéré, puis comparer les nouveaux états avec chacun des états déjà construits. La complexité est inférieure à $O(\frac{N_{au}^2 \cdot |\Sigma|}{2})$.

En fait, il y a une grande différence entre l'automate associé à l'algorithme de Needleman & Wunsch, et celui de la recherche de la meilleure occurrence d'un mot au sens Needleman & Wunsch. Lorsqu'on regarde la première coordonnée des vecteurs de ce second algorithme, on se rend compte qu'elle vaut toujours 0 : on ne tient pas compte de la pénalité des insertions/délétions due à la position de la meilleure occurrence du mot recherché dans la séquence *cible*[1]. Pour calculer l'automate, on n'a pas à tester la colinéarité (au sens $(max, +)$) du vecteur courant et des autres précédemment calculés, mais uniquement leur égalité. Informatiquement, c'est plus simple, même si la condition de colinéarité est moins contraignante.

**Remarque 3.11.1 :** Déjà en 1985, Esko Ukkonen a envisagé ce type d'algorithmes [126]. Il s'agissait du problème de *pattern matching* avec erreurs. On se donne un mot $p$, une chaîne de caractères $x$ et un entier $t$. Le problème s'exprime ainsi : trouver si une sous-chaîne de $x$ est proche du mot $p$, au sens où cette sous-chaîne et le mot $p$ diffèrent de au plus $t$ erreurs. Une erreur est soit un

1. Lorsqu'on effectue la comparaison d'une séquence avec l'ensemble d'une banque de données, on recherche les similarités entre une seule séquence, appelée *séquence requête* et les séquences de la banque, appelées *séquences cibles*.

*appariement non-exact*, soit une *insertion*, soit une *délétion*. L'algorithme proposé consiste en la construction d'un *automate fini déterministe* qui reconnaît tout mot de $\Sigma^*$ et arrive en un état final uniquement si $x$ contient une chaîne dont la distance d'édition à $p$ est inférieure à $t$.

Chaque état correspond intuitivement à une ligne de la matrice de la programmation dynamique. Ukkonen propose de diminuer le nombre d'états, en regroupant les états qui correspondent à des lignes $S = (s_0, s_1, \ldots, s_m)$ et $R = (r_0, r_1, \ldots, r_m)$ si ces deux lignes diffèrent uniquement pour des coordonnées $i$ telles que : $s_i < -t$ et $r_i < -t$. En effet, dans la matrice construite par programmation dynamique, tout chemin passant par une cellule de score $< -t$, ne peut mener à une cellule de la dernière ligne ayant un score $> -t$.

**Remarque 3.11.2 :** L'algorithme de *pattern matching* sans erreur donné par Baeza-Yates & Gonnet [11] utilise aussi le principe de programmation dynamique (Cf. annexe B.2.1).

## 3.12 Résultats

Les résultats sont très satisfaisants pour de petits mots (longueur inférieure à 13). Le tableau 3.4 synthétise les gains de cette nouvelle implémentation de la programmation dynamique. Plusieurs remarques peuvent être faites à la vue de ces résultats.

1. Si l'automate est stocké, le temps de *databank scanning* est le temps de parcours de la banque (colonne (2)).

   Théoriquement le temps de parcours de la banque est constant. Les variations qu'on observe proviennent de la gestion de la mémoire. Plus l'automate est gros, plus nombreux seront les déplacements dans la mémoire.

2. Le temps nécessaire à la première phase du calcul, à savoir le calcul de l'automate, n'est pas constant. On observe une explosion du temps nécessaire à la construction de l'automate, due à une explosion de la taille de l'automate.

3. Dans l'implémentation actuelle, il y a une taille optimale pour l'utilisation de cet algorithme. Lorsqu'on regarde la différence entre le temps d'exécution de l'algorithme de programmation dynamique et celui de l'algorithme avec automate, le gain est maximal pour une séquence de longueur 12/13.

4. La taille de l'automate semble être une fonction exponentielle de la longueur de la séquence. Mais elle dépend aussi du nombre de générateurs du semi-groupe de matrices. Plus il y a de lettres différentes dans la séquence *requête*, plus nombreux seront les générateurs. La séquence BAAABF est composée de trois lettres différentes. Le nombre de générateurs est 4, puisque les matrices génératrices associées à deux lettres différentes non-présentes dans la séquence BAAABF, seront identiques.

   Cette remarque n'est plus valable lorsque la matrice de similarité est quelconque. Dans ce cas, le nombre de générateurs est au maximum égal à la taille de l'alphabet.

5. Cet algorithme peut être amélioré. L'implémentation actuelle sépare complètement la phase de construction de l'automate de la phase de parcours de la banque de données. On peut envisager de construire les états de l'automate au fur et à mesure qu'on en a besoin. Certains états peuvent ne jamais être utilisés, et dans de tels cas, leur calcul est inutile. Le temps global d'exécution ne devrait pas être augmenté.

Cet algorithme est intéressant si le mot recherché est de petite taille ($|m| \leq 13$), et si la banque est très grande. Le temps nécessaire à la construction de l'automate doit être réparti sur un nombre important de séquences *cibles*. Au fur et à mesure que les banques grandissent, la taille des séquences *requêtes* peut augmenter tout en gagnant en temps de calcul.

## 3.13 Développements : éléments probabilistes

Considérons le problème de la recherche de la probabilité d'avoir un score donné. Les travaux sur ce point sont nombreux, on pourra se référer à la partie 5 pour avoir une idée des développements scientifiques sur ce sujet. Le formalisme de l'automate permet une approche différente. L'automate associé à une séquence *requête* va permettre de calculer pour cette séquence la distribution du score d'alignement de cette séquence contre une séquence de longueur donnée.

**Modèle M0 :** Considérons une séquence aléatoire sous le modèle **M0**. Chaque lettre est indépendante des autres, et la probabilité d'avoir en une place donnée, une lettre spécifique de l'alphabet est donnée a priori.

On peut essayer de calculer la probabilité d'être dans un état $j$ de l'automate au temps $n$. Pour cela, on associe à chaque transition de l'automate une probabilité, celle d'apparition de la lettre associée à cet arc. L'automate devient simplement une chaîne de Markov à $T$ états. On sait alors calculer la distribution des états au temps $n$. Soit $M$ la matrice de transition. Puisque de chaque état ne partent que $|\Sigma|$ arcs et que $|\Sigma| \ll T$, la matrice $M$ est creuse. Soit $E_n$ le vecteur de distribution des états au temps $n$. $E_n$ s'obtient par la formule suivante :

$$E_n = E_0.M^n \tag{3.44}$$

$$E_0 = (1, 0, \ldots, 0) \tag{3.45}$$

A partir de la distribution des états au temps $n$, on peut calculer la distribution des scores pour la comparaison de la séquence *requête* avec une séquence aléatoire de longueur $n$ sous le modèle M0. En effet, à chaque état est associé un score. Si on somme les probabilités d'être dans un état $j$, sur l'ensemble des états qui ont le même score $s$, on obtient la probabilité d'avoir un score égal à $s$ au temps $n$.

D'autre part, si $(M^n)_{n\in\mathbb{N}}$ converge, la suite $(E_n)_{n\in\mathbb{N}}$ converge vers la loi limite qui donne la distribution des états à temps infini. La distribution des scores à temps infini peut être générée de la même manière.

Cette méthode n'est valable que si tous les coefficients de proportionnalité, au sens $(max, +)$, associés aux transitions sont nuls. En effet, le vecteur de distribution des états au temps $n$ d'une chaîne de Markov ne tient pas compte du chemin menant à cet état. Or le score associé à un état dépend du chemin qui a permis d'atteindre cet état. Il faudrait pouvoir, pour passer à la distribution des scores au temps $n$, distinguer les chemins de coûts différents.
Cette méthode n'est donc pas applicable à l'algorithme Needleman & Wunsch.

**Modèle M1 :** Considérons une séquence aléatoire sous le modèle **M1** : on connaît les lois de transition, la probabilité d'avoir la lettre $b$ après la lettre $a$ est donnée par :

$$P(b|a) = p_{ab} \quad \forall a, b \in \Sigma.$$

On peut se ramener au cas d'une chaîne de Markov simple, si on multiplie par la taille de l'alphabet le nombre d'états de l'automate. En effet, la topologie de l'automate ne change pas, mais il faut stocker la lettre qui a permis de se retrouver dans l'état considéré de l'automate.

| Mots recherchés | long. | Programmation Dynamique | | Automate | | | |
|---|---|---|---|---|---|---|---|
| | | temps (en s.) | vitesse [a] | temps (en s.) total | (1) [b] | (2) [c] | taille de l'automate |
| BAAABF | 6 | 15.20 | 8.37 | 9.795 | 0.005 | 9.79 | 83 |
| KIIKLHEN | 8 | 19.93 | 8.51 | 9.86 | 0.034 | 9.826 | 472 |
| VKIIKLHEN | 9 | 22.25 | 8.58 | 10.284 | 0.087 | 10.197 | 1114 |
| AASDTGSTYL | 10 | 24.48 | 8.66 | 10.442 | 0.18 | 10.262 | 2397 |
| LVIVSVFDLAS | 11 | 26.68 | 8.74 | 10.885 | 0.405 | 10.48 | 4928 |
| KNVIGARRASWR | 12 | 28.69 | 8.87 | 12.176 | 1.237 | 10.939 | 12033 |
| RAANQDYVITRTN | 13 | 31.11 | 8.86 | 13.935 | 2.716 | 11.219 | 24331 |
| QGQQFPNECQLDQL | 14 | 33.16 | 8.95 | 17.389 | 6.089 | 11.30 | 50820 |
| QGQQFPNECQLDQLN | 15 | 35.49 | 8.96 | 25.916 | 13.098 | 12.818 | 107408 |

TAB. 3.4 - **Performances comparées de la Programmation Dynamique simple et de l'algorithme dérivant de l'écriture $(max,+)$ :** *Pour chaque mot considéré, on recherche dans chaque séquence de SwissProt Rel. 34, la position de la sous-séquence qui, alignée avec le mot considéré, minimise le nombre d'erreurs. On compare les performances des algorithmes classiques de programmation dynamique et de l'algorithme dérivant de la formulation $(max,+)$. Les deux algorithmes sont implémentés dans LASSAP [48]. Les résultats sont ceux obtenus sur serveur SUN 4000, en utilisant un seul processeur. Les vitesses représentent le ratio MMC/s (million matrix cells per seconds).*

[a] (en MMC/s)
[b] Construction de l'automate
[c] Parcours de la banque de séquences

# Chapitre 4

# L'algorithme de Smith & Waterman avec Motifs

*La comparaison de séquences a pour objectif d'essayer d'induire des connaissances sur une séquence inconnue en la comparant à une grande base de données. L'alignement de deux séquences permet de visualiser les différentes zones de similarité communes aux deux séquences.*

*Les différentes méthodes d'alignement existantes sont fondées sur des coûts d'édition calculés additivement, position par position, en fonction d'une matrice de substitution fixe. Ceci implique qu'une similitude a toujours le même poids, indépendamment de sa position. Or lorsque le biologiste réalise un alignement «à la main», il privilégie certaines similitudes en fonction de ses connaissances sur les structures ou fonctions des séquences.*

*Nous présentons ici une méthode d'alignement plus sensibles et plus précises, dans le sens où la méthode prend en compte la non-homogénéité des séquences. Cette hétérogénéité provient d'informations autres que la seule structure primaire. Une des informations que la méthode peut prendre en compte, provient du comportement du biologiste lorsqu'il aligne deux séquences. Il cherche à mettre en correspondance les motifs qu'il sait être pertinents pour les séquences considérées.*

*La méthode proposée vise à prendre en compte des informations liées à des* «motifs» *biologiques. Typiquement on pense à des séquences protéiques et à l'utilisation de motifs décrits dans la base* ***PROSITE****. Il ne s'agit pas d'****imposer*** *l'alignement de ces motifs car dans le cas où ces motifs apparaîtraient dans un ordre différent sur les deux séquences, on aboutirait à une impossibilité. Si les deux séquences impliquées sont reliées biologiquement, on est en face d'un processus d'inversion. Puisque les algorithmes classiques d'alignement de séquences ne gèrent pas les inversions, aucune solution alignant les deux motifs, ne sera exhibée. On peut citer cependant certains développements sur des alignements avec inversions [68].*

*La méthode consiste à réaliser un alignement par programmation dynamique lettre à lettre au sens de Smith & Waterman, en attribuant un bonus supplémentaire à tout chemin mettant en correspondance le même motif sur les deux séquences. On favorise ainsi les alignements dans lesquels les motifs sont alignés.*

*La première phase consiste à repérer les occurrences des motifs de la banque de données sur les deux séquences à l'aide d'un algorithme de «pattern matching». Après avoir repéré toutes les occurrences des motifs communs aux deux séquences, la deuxième étape met en œuvre la programmation dynamique pour laquelle l'espace des décisions a été élargi : pour tout couple d'indices $(i, j)$ qui correspond à la fin de deux réalisations d'un même motif dans chacune des deux séquences, on rajoute la possibilité d'aligner les deux motifs, en pondérant cet alignement.*

*Cet algorithme a été testé sur l'ensemble des motifs **PROSITE** présents dans la banque de protéines SwissProt Rel. 35. Lorsque les deux séquences partagent plusieurs motifs, l'algorithme permet le raccordement de deux zones de similarité très éloignées.*

## 4.1 La banque *PROSITE*

La banque de données ***PROSITE*** est une banque de motifs (patterns) déterminés par des expressions régulières. Dans cette banque on trouve d'autres types de caractérisation de fonction biologique (*Matrix, Rules*), mais on n'en tiendra pas compte ici. Pour l'instant, on ne retiendra que les motifs de la banque qui sont au nombre de 1275 dans ***PROSITE*** Rel. 14.0. Il y a 56 Matrix et 4 Rules. Pour des raisons de taille, quelques-uns des motifs, les plus représentés dans la banque SwissProt Rel. 35, ont été éliminés :

| Motifs PROSITE | Nombres d'occurrences dans SwissProt | Expressions régulières associées |
|---|---|---|
| PS00001 | 141147 | N-P-[ST]-P |
| PS00004 | 42117 | [RK](2)-x-[ST] |
| PS00005 | 332212 | [ST]-x-[RK] |
| PS00006 | 377147 | [ST]-x(2)-[DE] |
| PS00008 | 372978 | G-{EDRKHPFYW}-x(2)-[STAGCN]-{P} |
| PS00009 | 23386 | x-G-[RK]-[RK] |
| PS00013 | 7969 | {DERK}(6)-[LIVMFWSTAG](2)-[LIVMFYSTAGCQ]-[AGS]-C |
| PS00016 | 4152 | R-G-D |
| PS00029 | 4007 | L-x(6)-L-x(6)-L-x(6)-L |

Ces motifs ne sont pas très informatifs. Certains sont très courts et apparaissent souvent sans que la fonction biologique associée soit présente. Une fois ces motifs enlevés, la banque contient 1266 motifs.

Quelques outils performants permettent de rechercher si un motif connu de la banque ***PROSITE*** est contenu dans une séquence, ou inversement, permettent de rechercher dans une banque de séquences celles qui contiennent un motif particulier.

Ces motifs servent à caractériser une propriété commune à toute une famille de protéines. Malheureusement, un motif peut être syntaxiquement présent dans une séquence mais ne pas partager la propriété biologique associée. Les séquences qui partagent l'expression régulière, mais non la propriété biologique, sont appelées *faux positifs*. Les faux positifs de la banque SWISSPROT sont indexés dans la banque ***PROSITE***. De la même manière, certaines séquences partagent la propriété biologique mais pas l'expression régulière.

Les patterns ***PROSITE*** ressemblent aux patterns UNIX. La grammaire des expressions régulières sous UNIX est simple.

- Tout caractère représente une occurrence de ce caractère.
- Le symbole «.» représente un caractère quelconque.
- Le *ou* logique s'écrit «|».
- L'opérateur de closure «*» représente une suite d'occurrences de l'élément sur lequel elle s'applique, l'opérateur «+» représente une suite non nulle d'occurrences, et «?» représente soit la chaîne de caractères nulle, soit une seule occurrence de l'élément sur lequel il s'applique.
- Les symboles «^» et «$» représentent respectivement le début et la fin de ligne.

– «[...]» signifie n'importe quel caractère parmi ceux contenus dans les crochets.

– «[^...]» signifie n'importe quel caractère sauf ceux contenus dans les crochets.

Par exemple l'expression «^[^A][BC]A*B.» décrit les chaînes de caractères se trouvant en début de ligne, dont la première lettre n'est pas «A», suivi par «B» ou «C», puis par un nombre quelconque de «A», et terminant par «B» puis n'importe quelle autre lettre. La figure 4.1 donne un exemple d'un motif ***PROSITE***.

```
ID   PROTEIN_SPLICING; PATTERN.
AC   PS00881;
DT   OCT-1993 (CREATED); NOV-1995 (DATA UPDATE); NOV-1995 (INFO UPDATE).
DE   Protein splicing signature.
PA   [LIVA]-[LIVMY]-[VAT]-H-N-[STC].
NR   /RELEASE=32,49340;
NR   /TOTAL=22(21); /POSITIVE=6(5); /UNKNOWN=0(0); /FALSE_POS=16(16);
NR   /FALSE_NEG=0; /PARTIAL=0;
CC   /TAXO-RANGE=A?EP?; /MAX-REPEAT=2;
DR   P30317, DPOL_THELI, T; P35901, RECA_MYCLE, T; P26345, RECA_MYCTU, T;
DR   P38078, VATA_CANTR, T; P17255, VATA_YEAST, T;
DR   P34914, HYES_MOUSE, F; P34784, PHEG_AGLNE, F; Q06927, VBR1_PHUV , F;
DR   P03564, VBR1_TGMV , F; Q06661, VBR1_TMOV , F; P42702, LIFR_HUMAN, F;
DR   P18816, LACR_LACLA, F; P38309, YB56_YEAST, F; P38626, NC5R_YEAST, F;
DR   P03905, NU4M_HUMAN, F; P19809, EAE1_ECOLI, F; P24719, MEK1_YEAST, F;
DR   Q06805, TIE1_BOVIN, F; P35590, TIE1_HUMAN, F; Q06806, TIE1_MOUSE, F;
DR   P03317, POLN_SINDV, F;
DO   PDOC00687;
```

FIG. 4.1 - **Une entrée de la banque *PROSITE***

Pour la reconnaissance des motifs dans les séquences, on peut transcrire le motif ***PROSITE*** avec la grammaire des motifs UNIX, puis utiliser les algorithmes classiques de recherche d'expressions régulières : Regexp sous UNIX.

## 4.2 L'algorithme Smith & Waterman aligne-t-il les motifs ?

Puisqu'on cherche à améliorer l'alignement de séquences en prenant en compte l'information apportée par les motifs ***PROSITE***, il semble naturel de vérifier que l'alignement habituel ne met pas en correspondance les réalisations d'un même motif, présentes dans chacune des deux séquences. La programmation dynamique aligne-t-elle les motifs?

Dans la banque ***PROSITE***, chaque motif renvoie aux séquences de SwissProt qui contiennent ce motif (champs DR pour Data bank Reference - Cf. figure 4.1). On distingue 5 types de séquences contenant le motif en question :

1. Les **vrais positifs** (annotés dans le champ DR par **T**) sont les séquences contenant l'expression régulière modélisant le motif, et qui appartiennent à la famille biologique définie par le motif (Cf. figure 4.1).

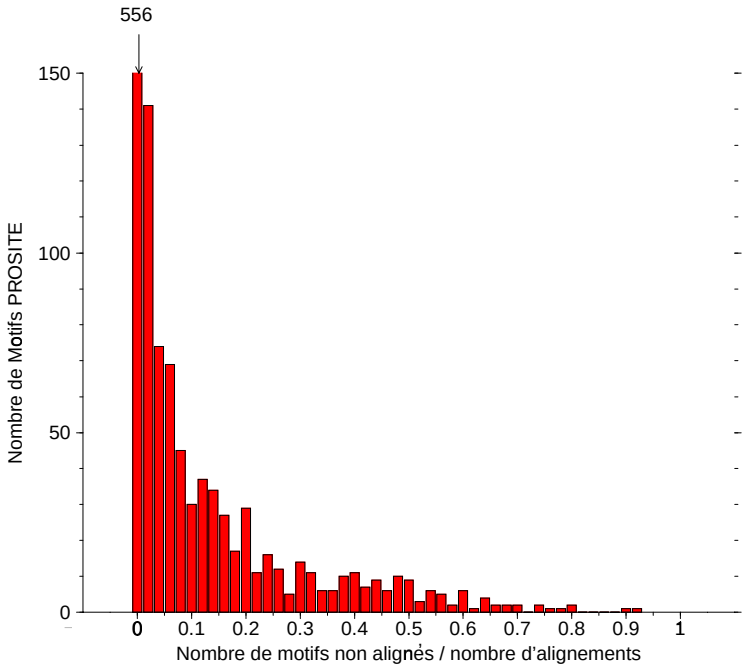



FIG. 4.2 - **Répartition des motifs de PROSITE Rel. 14.0 (vrais positifs)** *en fonction du nombre de motifs non alignés par la programmation dynamique. La banque de protéines utilisées est SwissProt Rel. 35. A la valeur* 0 *en abscisse, on observe* 556 *motifs en ordonnée. Pour ces 556 motifs, la programmation dynamique classique aligne par défaut les deux réalisations de ces motifs lorsque les deux séquences sont des* vrais positifs.

2. Les **faux négatifs** (annotés par **N**) sont les séquences qui appartiennent à la famille, mais qui ne contiennent pas le motif. On appellera **vrais négatifs** les séquences qui ne contiennent pas le motif et qui n'appartiennent pas à la famille considérée.

3. Les **faux positifs** (annotés par **F**) sont les séquences contenant le motif, mais qui n'appartiennent pas à l'ensemble considéré (Cf. figure 4.1).

4. Les sites **potentiels** (annotés par **P**) représentent les séquences qui appartiennent à l'ensemble considéré, mais qui n'ont pas le motif, parce que la région qui contient l'empreinte n'est pas encore accessible dans la banque.

5. Les sites **inconnus** (annotés par le symbole « **?**») sont les séquences dont on ne sait pas encore si elles appartiennent à l'ensemble considéré.

### 4.2.1 Les motifs et les vrais positifs

Pour les vrais positifs, on peut regarder si la programmation dynamique aligne par défaut la zone du motif. Si c'est le cas, le motif n'est pas vraiment informatif, puisque la programmation dynamique classique met en correspondance les deux réalisations de l'expression régulière. Aucune information supplémentaire n'est engendrée par la connaissance de l'expression régulière.

La figure 4.2 montre la répartition des motifs de la banque ***PROSITE*** Rel. 35 en fonction de la proportion du nombre de motifs non alignés par rapport à l'ensemble des alignements possibles entre deux vrais positifs. Autrement dit, pour chaque motif de la banque :

- on calcule l'alignement SW pour toutes les paires de vrais positifs,
- on regarde si l'alignement SW met en correspondance les deux réalisations de l'expression régulière,
- on comptabilise tous les alignements qui n'alignent pas les 2 réalisations du motif,
- on fait le ratio de ce nombre par le nombre d'alignements possibles et on obtient l'indice $p$.

Plus l'indice $p$ est proche de 0, plus nombreuses sont les réalisations alignées de l'expression régulière. Si $p = 0$, pour toute paire de séquences ayant le motif, la programmation dynamique classique aligne les occurrences du motif. A l'inverse, si $p = 1$, pour toute paire de séquences ayant le motif, l'algorithme SW n'aligne pas les réalisations du motif.

L'algorithme SW aligne les réalisations des motifs lorsque les réalisations sont longues et similaires ou lorsque elles sont très proches. L'algorithme SW ne les aligne pas lorsque les réalisations des motifs de sont pas similaires, ou lorsque elles sont très courtes et n'appartiennent pas à la zone de plus forte similitude.

On remarque que pour beaucoup de motifs, la programmation dynamique classique permet de mettre en évidence la zone correspondant au motif. Il y a 556 motifs pour lesquels la programmation dynamique classique aligne dans tous les cas le motif. Il s'agit vraisemblablement de motifs pas très informatifs puisque l'alignement regroupe instinctivement ces zones. En fait, le motif est soit très restrictif, (un mot uniquement), soit la réalisation du motif est placée dans une zone de forte similarité. Dans ce dernier cas, on peut se demander si le motif est bien défini. En effet puisque dans tous les cas la zone de similarité est plus large que le motif, il est peut-être possible d'allonger la définition de l'expression régulière.

Dans d'autres cas, la programmation dynamique passe à côté du motif. Pourtant, pour tous les motifs de SwissProt Rel. 35, la programmation dynamique aligne, au moins dans un cas, les réalisations du motif. Autrement dit, on ne trouve pas un seul motif pour lequel l'indice $p$ vaut 1. Pour un indice $p$ proche de 1, l'histogramme de la figure 4.2 a des classes totalement vides.

**Remarque :** Dans la majorité des cas, la programmation dynamique aligne les motifs, surtout lorsque les deux réalisations de motifs sont longues et/ou très similaires. Par contre, dans les autres cas, l'algorithme prenant en compte l'information due à la présence des motifs, semble pouvoir améliorer les alignements.

### 4.2.2 Les motifs et les faux positifs

**Faux positifs contre faux positifs :** On regarde, de même, si la programmation dynamique aligne les motifs ***PROSITE*** dans le cas des faux positifs. La figure 4.3 montre que la répartition ne se fait absolument pas de la même manière. La programmation dynamique classique a tendance à ne pas aligner les motifs.

**Faux positifs contre les vrais positifs :** Pour ce cas de figure, pour chaque motifs, on calcule l'alignement SW pour toutes les paires possibles de séquences telles que l'une des séquence est un faux positif, et l'autre un vrai positif. L'indice $p$ est ensuite calculé de la même manière. On

Figure A

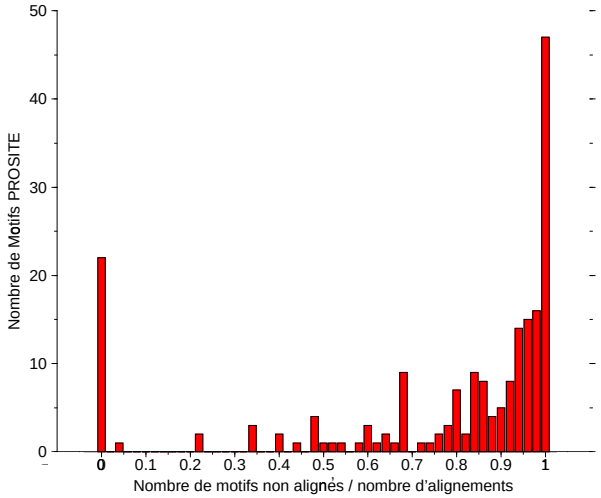



Figure B

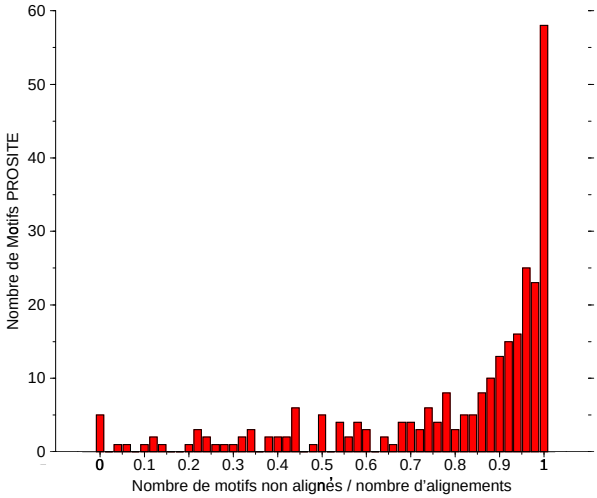



FIG. 4.3 - **Répartition des motifs de PROSITE Rel. 14.0 (faux positifs)** *en fonction du nombre de motifs non alignés par la programmation dynamique.*
**Figure A :** *On regarde si la programmation dynamique aligne deux occurrences du même motif PROSITE, pour des faux positifs.*
**Figure B :** *On regarde si la programmation dynamique aligne une occurrence du motif PROSITE (vrai positif) avec une occurrence du même motif pour un faux positif. On peut comparer ces deux figures avec la figure 4.2.*

retrouve le même comportement de l'algorithme de programmation dynamique, que dans le cas de l'alignement de deux faux positifs. L'algorithme a plutôt tendance à ne pas aligner les deux occurrences du même motif (Cf. figure 4.3).

Dans les deux cas, on observe un pic pour la valeur $p = 0$. Le pic est beaucoup plus important dans le graphique portant sur la comparaison des faux positifs entre eux. Revenons à la figure 4.1, qui décrit une entrée ***PROSITE***. Parmi les faux positifs, on trouve les séquences : TIE1_BOVIN, TIE1_HUMAN et TIE1_MOUSE. Rien qu'à la vue des noms des séquences, on peut supposer que ces trois protéines appartiennent à une même famille. Vérification faite, les séquences sont toutes les trois des «*TYROSINE-PROTEIN KINASE RECEPTOR TIE-1 PRECURSOR*». Lorsque ces trois séquences sont alignées, la quasi-totalité des séquences appartiennent à la zone de plus grande similitude.

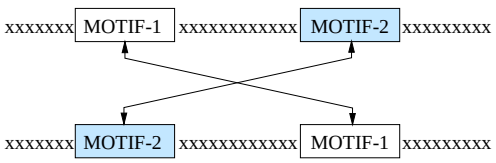
Cet exemple illustre qu'on retrouve parmi les faux positifs d'un motif, des séquences appartenant à une même famille. Si le motif est à l'intérieur de la zone de haute similarité définissant la famille, l'algorithme SW aligne les réalisations du motif. Lorsque tous les faux positifs d'un motif appartiennent à la même famille, l'indice $p$ vaut 0. Ce phénomène n'apparaît pas lors de la comparaison d'un faux positif avec un vrai positif.

**Hypothèse :** Il serait intéressant de vérifier l'hypothèse suivante. Les occurrences d'un même motif sont alignées dans le cas des vrais positifs, parce que le motif est contenu dans une zone de *grande similarité*. Dans le cas des faux positifs, les occurrences de ces motifs n'apparaissent pas dans des zones de similarité. Cette hypothèse est à mettre en relation avec la méthode de construction de la banque de données ***PROSITE***. La banque est construite à partir d'alignements multiples. Les zones qui permettent la détermination de l'expression régulière qui définit le motif, sont des zones de grande similarité.

## 4.3 Algorithme Smith-Waterman-Motif : SWM

L'algorithme de Smith & Waterman ne met toujours en correspondance les occurrences du même motif dans les deux séquences à aligner. L'alignement des réalisations des motifs peut alors être un guide pour améliorer l'alignement des séquences. Une meilleure coïncidence des réalisations des motifs est recherchée. Il s'agit alors d'une solution intermédiaire entre l'algorithme SW et la méthode utilisée par les biologiste qui consiste à imposer l'alignement des motifs.

Le but du nouvel algorithme est de favoriser les alignements qui mettent en correspondance deux occurrences du même motif. Si les deux séquences ne partagent qu'un seul motif, ou si on cherche à n'aligner qu'un seul motif, il est envisageable d'imposer l'alignement du motif, puis d'utiliser deux fois l'algorithme de Smith & Waterman pour étendre l'alignement de part et d'autre. Cependant, si les deux séquences partagent plusieurs motifs qui peuvent apparaître dans des ordres différents, on ne peut plus imposer l'alignement des motifs. On arriverait à des impossibilités. Supposons que les deux séquences soient du type :




Si les deux séquences sont biologiquement reliées, on se trouve face à une inversion. Les algorithmes

classiques sont incapables d'aligner les deux motifs simultanément. Notons que notre algorithme ne résout pas ce problème, mais choisit lequel des deux motifs (*MOTIF-1* ou *MOTIF-2*) sera aligné.

La première étape est de déterminer tous les motifs que les deux séquences ont en commun. Pour ce faire, on peut directement utiliser le logiciel GCG, avec la fonction *motifs*, qui détermine la liste de tous les motifs présents dans la séquence considérée. Mais, dans notre cas, il faut déterminer les motifs présents à la fois dans les deux séquences. Un recoupement des deux listes de motifs présents dans chacune des deux séquences, est nécessaire.

Notons aussi, que pour la majorité des motifs de la banque ***PROSITE***, la banque donne la liste de toutes les séquences de SwissProt Rel. 35 contenant ces motifs. Ces références croisées peuvent permettre d'accélérer cette phase préliminaire.

### 4.3.1 Alignement de deux réalisations du même motif

Un premier problème est à soulever : un motif PROSITE peut contenir des insertions/délétions. Prenons le cas où l'expression régulière définissant le motif contient un terme du type «$x\{6,8\}$». A cet endroit, on doit trouver une chaîne non déterminée d'une longueur comprise entre 6 et 8. Deux réalisations du même motif peuvent avoir deux longueurs différentes. Dans ce cas, l'alignement devra faire apparaître une insertion/délétion à une place bien particulière : à l'intérieur des deux chaînes de caractères correspondant à «$x\{6,8\}$».

En revanche, dans le cas des motifs *à longueur constante*, l'alignement est simple, puisqu'il ne peut y avoir d'insertions/délétions. Prenons l'exemple du motif Asp_Protéase dont l'expression régulière est :

```
[LIVMFGAC]-[LIVMTADN]-[LIVFSA]-D-[ST]-G-[STAV]-[STAPDENQ]-x-[LIVMFSTNC]-x-[LIVMFGTA]
```

On le trouve dans les séquences bar1_yeast et tryp_astfl. Il apparaît deux fois dans la séquence bar1_yeast en position 60 et 284, et une fois dans tryp_astfl en position 194.

```
bar1_yeast:                (V)(L)(F)D(T)G(S)(A)x(F)x(V)
             60:   SQSLT    V  L  F D T G S  A D F W V     MDSSN

                           (V)(L)(L)D(S)G(T)(S)x(L)x(A)
             284   TTKYP    V  L  L D S G T  S L L N A     PKVIA

tryp_astfl:                (A)(A)(S)D(T)G(S)(T)x(L)x(G)
             194   SGGPL    A  A  S D T G S  T Y L A G     IVSWG
```

L'alignement de la première occurrence du motif dans bar1_yeast avec la seule occurrence dans tryp_astfl sera :

```
 60     V L F D T G S A D F W V
              | | | |
194     A A S D T G S T Y L A G
```

Si on décide d'aligner la deuxième occurrence du motif dans bar1_yeast, on obtientra :

```
284     V L L D S G T S L L N A
              |   |       |
194     A A S D T G S T Y L A G
```

Le score d'alignement est la somme des coûts de substitutions :

$$Score(Motif_A, Motif_B) = \sum_{k=0}^{long-1} S\Big(A[\beta_A + k], A[\beta_B + k]\Big)$$

où

- $Motif_A$ et $Motif_B$ sont les motifs dans les séquences A et B,
- $\beta_A$ et $\beta_B$ sont les indices de début du motif,
- *long* est la longueur commune des deux motifs.

Le problème de l'alignement devient subtil lorsque la longueur du motif partagé par les deux séquences n'est pas constante. Exemple: le motif Snake_Toxin se présente sous la forme suivante:

```
CPx{6,8}(L,I,V,Y,S,T)xCC
```

On le trouve dans les séquences hcy_octdo et nxl2_bunfl:

```
hcy_octdo :                CP  x{6}  (Y)xCC
             468   HGSTL   CP SPEEPK  Y ACC     LHGMP
nxl2_bunfl:                CP  x{8}  (L)xCC
              46   GCAAT   CPEFTSRYKS L LCC     TTDNC
```

Dans ce cas, on ne peut pas simplement appliquer l'algorithme de Needleman & Wunsch, puisqu'il y a une insertion/délétion à une place bien déterminée. L'algorithme NW ne garantit pas que l'insertion/délétion intervienne dans la région à longueur variable. Par contre, l'alignement a un sens si on cherche à aligner en priorité les symboles «(», «)», «{» et «}». Alignons par exemple les séquences:

```
{CP{SPEEPK}(Y)A(CC)}
```

et

```
{CP{EFTSRYKS}(L)L(CC)}
```

sur un alphabet de 24 lettres. On donne sur cet alphabet une nouvelle matrice de substitution:

- L'élément de la matrice correspondant à un couple de la forme $\left(\text{«\{»},\alpha\right)$,$\left(\text{«(»},\alpha\right)$, $\left(\text{«}\}\text{»},\alpha\right)$, $\left(\text{«)»} \text{ et } \alpha\right)$ où $\alpha$ est une lettre de l'alphabet de 20 lettres représentant les 20 acides aminés, est arbitrairement pénalisant.
- L'élément de la matrice correspondant à $\left(\text{«\{»},\text{«\{»}\right)$, $\left(\text{«}\}\text{»},\text{«}\}\text{»}\right)$, $\left(\text{«(»},\text{«(»}\right)$ ou à $\left(\text{«)»},\text{«)»}\right)$ est arbitrairement grand.
- Les éléments $\left(\text{«\{»},\text{«( »}\right)$, $\left(\text{«\{»},\text{«)»}\right)$, $\left(\text{«\{»},\text{«}\}\text{»}\right)$, $\left(\text{«}\}\text{»},\text{«(»}\right)$, $\left(\text{«}\}\text{»},\text{« )»}\right)$, et $\left(\text{«(»},\text{«)»}\right)$ valent -$\infty$.

On peut utiliser l'algorithme de Needleman & Wunsch sur ce nouvel alphabet. On ne retiendra que l'alignement concernant les lettres des acides aminés. On obtient:

```
{CP{--SPEEPK}(Y)A(CC)}          CP--SPEEPKYACC
||||         || | |||||  Soit    ||           ||
{CP{EFTSRYKS}(L)L(CC)}          CPEFTSRYKSLLCC
```

Le score de l'alignement est le score trouvé par l'algorithme de Needleman & Wunsch, auquel on aura retranché les coûts liés à l'alignement des 4 lettres supplémentaires «{», «}», «(», et «)».

**Une meilleure solution**

Lors de la recherche des réalisations du motif dans les deux séquences, on peut rechercher les positions possibles des insertions/délétions. Par exemple, si on trouve le motif Snake_Toxin sous la forme CPSPEEPKYACC, les éventuelles insertions/délétions lors de l'alignement avec une autre réalisation de ce motif auront lieu dans la zone *SPEEPK*, et uniquement dans cette zone. Lorsqu'on cherche à aligner deux réalisations d'un même motif, on effectue les différentes phases suivantes :

1. Recherche des positions possibles des insertions/délétions sous forme d'un tableau :

| Indice du tableau | Position de Début | Position de fin |
|---|---|---|
| i | $b_i$ | $e_i$ |

où $(b_i, e_i)$ définissent la *plage d'insertion/délétion* numéro $i$. S'il y a une insertion/délétion, elle doit obligatoirement se trouver dans une des plages définies par le tableau précédent. On a deux tableaux, l'un pour la séquence $A$ (indices $(b_i^A, e_i^A)$) et l'autre pour $B$ (indices $(b_i^B, e_i^B)$).

2. Alignement des deux occurrences du motif: posons $(i, j)$ l'indice courant de l'alignement.
   - Si $(i, j)$ n'appartient à aucune plage d'insertion/délétion, on aligne la lettre $A_i$ avec la lettre $B_j$, et on passe à l'indice $(i + 1, j + 1)$.
   - Si $(i, j)$ appartient à la plage numéro $k$, on lance un alignement NW sur les sous-séquences $A[b_k^A, e_k^A]$ et $B[b_k^B, e_k^B]$, puis on passe à l'indice $(e_k^A + 1, e_k^B + 1)$.

### 4.3.2 Alignement local en tenant compte des motifs

Maintenant qu'on a donné un sens à un alignement de motifs, on peut essayer d'introduire un alignement tenant compte des motifs contenus dans les deux séquences. Pour des raisons de simplicité, nous ne considèrerons que le cas des pénalités linéaires des insertions/délétions ($go = ge = \delta$). Dans le cas des pénalités affines, la modification de l'algorithme est strictement la même. Utiliser des pénalités affines ne ferait que compliquer l'exposé, sans apporter d'éléments nouveaux.

Dans l'algorithme de Smith & Waterman, lorsqu'on calcule l'élément $(i, j)$ de la matrice de comparaison, on détermine le

$$\max \begin{pmatrix} M(i-1,j) - \delta, \\ M(i-1,j-1) + S(i,j), \\ M(i,j-1) - \delta, \\ 0 \end{pmatrix}.$$

Supposons maintenant que la séquence A contienne un motif en position $(i - long_1 + 1, i)$ et que la séquence B contienne le même motif en position $(j - long_2 + 1, j)$, où $long_1$ et $long_2$ sont les longueurs des l'occurrences du motif dans les séquences A et B. Alors, on peut considérer un chemin possible en plus de la diagonale, la verticale et l'horizontale : l'alignement du motif. Ainsi on détermine le

$$\max \begin{pmatrix} M(i-1,j) - \delta, \\ M(i-1,j-1) + S(i,j), \\ M(i,j-1) - \delta, \\ 0, \\ M(i - long_1, j - long_2) + Score(motif_1, motif_2) + Bonus \end{pmatrix} \quad (4.1)$$

avec les notations :

- $Score(motif_1, motif_2)$ est le score obtenu pour l'alignement des deux occurrences du motif dans les deux séquences,
- $Bonus$ est la valeur positive pondérant l'alignement du motif,
- $M(k,l)$ est le score obtenu pour l'alignement des sous-séquences $A_1A_2...A_k$ et $B_1B_2...B_l$.

**Remarque :** Cette modification de la programmation dynamique entraîne une augmentation de l'ordre de l'algorithme. La version habituelle de la programmation dynamique fait dépendre chaque élément $(i,j)$ de ses proches voisins : $(i-1,j-1)$, $(i,j-1)$ et $(i-1,j)$. Lorsque les pénalités des insertions/délétions sont affines, il faut prendre en compte deux autres valeurs.

Ici, la dépendance de l'élément $(i,j)$ est élargie lorsque cette position correspond à la fin de deux réalisations du même motif. La cellule $(i,j)$ dépend non seulement de ces 3 voisins immédiats mais aussi de la cellule correspondant aux débuts des motifs considérés. Cette cellule peut être loin de $(i,j)$ : il s'agit de l'élément $(i-long_1, j-long_2)$.

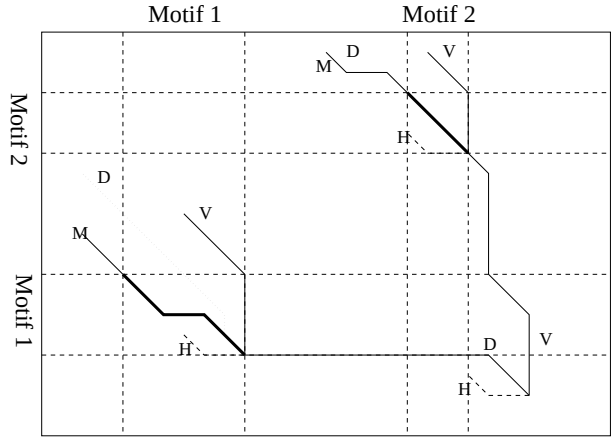



FIG. 4.4 - **Compétition entre deux motifs :** *Si les deux séquences partagent deux motifs incompatibles entre eux, l'algorithme choisit, en fonction des pondérations de chacun des deux motifs, d'aligner l'un ou l'autre, ou de ne pas en aligner.*

### 4.3.3 Algorithme SWM

Soient deux séquences $A$ et $B$, de longueurs respectives $n$ et $m$. Nous appellerons *motif* un couple de sous-séquences de type $(mot_A, mot_B)$, l'une appartenant à la séquence $A$ et l'autre à $B$, telles que $mot_A$ et $mot_B$ sont des réalisations du même motif. Pour reprendre l'exemple précédent, $mot_A$ =CPSPEEPKYACC et $mot_B$ =CPEFTSRYKSLLCC.

Les deux séquences $A$ et $B$, peuvent partager plusieurs motifs. De plus, un même motif peut apparaître plusieurs fois dans la même séquence. Une position $(i,j)$, peut correspondre à la fin de plusieurs motifs. Si tel est le cas, le *maximum* doit être pris sur tous les alignements de motifs finissant en $(i,j)$.

La récurrence de l'algorithme SWM s'exprime ainsi :

$$M(i,j) = \max \left( \begin{array}{c} M(i-1,j-1) + S(A_i, B_j), \\ M(i-1,j) - \delta, \\ M(i,j-1) - \delta, \\ 0, \\ \max \left\{ \begin{array}{c} m, motif \\ \text{finissant en } (i,j) \end{array} \right\} \left( \begin{array}{l} M(a_1^m - 1, b_1^m - 1) + \\ Score(Motif_m^A, Motif_m^B) \\ \times Bonus(Motif_m) \end{array} \right) \end{array} \right) \tag{4.2}$$

où

- $Score(Motif_m^A, Motif_m^B)$ est le score d'alignement des deux réalisations ($motif_m^A$ et $Motif_m^B$) du même motif $m$,

- $Bonus(Motif_m)$ est un coefficient associé au motif $m$, pondérant l'alignement des réalisations de ce motif. Lorsque nous avons présenté l'algorithme, nous avons introduit un bonus additif (Cf. éq. 4.1).

Dans l'équation 4.2, le score d'alignement des motifs est *multiplié* par la valeur du $Bonus$. Plus les occurrences du motif sont proches, plus le score d'alignement est élevé. Ce modèle multiplicatif va alors préférer l'alignement des occurrences les plus proches. Soit deux séquences partageant chacune deux motifs (Cf. figure 4.4). Le modèle additif peut préférer aligner un motif dont les occurrences ne sont pas très proches, si l'alignement en dehors de ces motifs est élevé. Par contre le modèle multiplicatif préfèrera aligner des occurrences plus proches.

Pour toute la suite de ce chapitre, on ne retient que le modèle multiplicatif, qui va permettre, lorsqu'il y a concurrence entre plusieurs motifs, de pondérer de manière plus significative un alignement de motif dont les occurrences sont très proches (dont le score d'alignement est élevé).

### 4.3.4 Complexité de l'algorithme

L'algorithme se décompose en trois phases : la recherche de tous les motifs communs aux deux séquences, la programmation dynamique dont la récurrence principale est donnée par l'équation 4.2, et l'alignement des réalisations des motifs.

Pour une expression régulière donnée, la phase de recherche des occurrences du motif se fait en $O(m+n)$ [1], puisqu'on doit faire la recherche à la fois dans la séquence $A$ de longueur $m$ et dans la séquence $B$ de longueur $n$. Lorsqu'on prend en compte tous les motifs d'une banque de données, la complexité est augmentée d'un facteur égal à la taille de la banque.

La programmation dynamique a une complexité en $O(m \times n)$, même s'il y a une opération de maximisation supplémentaire en un petit nombre de cellules $(i,j)$.

Seule la complexité de l'alignement de deux réalisations du même motif nécessite un regard particulier.

- Dans le cas d'un motif de longueur fixe, l'algorithme est linéaire : il s'agit de faire la somme des coûts de substitution sur chacune des positions de l'alignement.

1. La complexité de l'algorithme de pattern matching est linéaire.

– Dans le cas d'un motif de taille variable, le calcul du score d'alignement nécessite la connaissance des positions des insertions/délétions éventuelles. Ensuite, on effectue autant de programmation dynamique NW qu'il y a de *plages d'insertions/délétions*. La complexité est alors

$$O\left(\sum_{i=1}^{Np}(e_i^A - b_i^A) \times (e_i^B - b_i^B)\right) + O\left(l_{m_A} + l_{m_B}\right)$$

où

– $Np$ est le nombre de plages d'insertions/délétions, définies par les indices $(b_i^A, e_i^A)_{i\in[1,Np]}$ et $(b_i^B, e_i^B)_{i\in[1,Np]}$,

– $l_{m_A}$ et $l_{m_B}$ sont les longueurs des réalisations du motif $m$ dans les séquences $A$ et $B$.

Le tableau 4.1 synthétise les différentes complexités intervenant au sein de l'algorithme.

| | **Complexité** |
|---|---|
| **Recherche des motifs en commun** | $O((n+m) \times t)$ |
| **Alignement d'un motif** | $O\left(l_{m_A} + l_{m_B} + \sum_{i=1}^{Np}(e_i^A - b_i^A) \times (e_i^B - b_i^B)\right)$ |
| **Programmation dynamique** | $O(m \times n)$ |

TAB. 4.1 - **Complexité de l'algorithme Smith-Waterman-Motif.**
*t est la taille de la banque de motifs.*

## 4.4 Influence de la pondération sur l'alignement

Globalement, on s'attend à aligner les occurrences des motifs de manière plus constante. La répartition des motifs de PROSITE suivant l'indice défini plus haut[2] (Cf. 4.2.1), est concentrée autour de l'indice 0, ce qui correspond à l'alignement des occurrences (Cf. figure 4.5). Lorsqu'on augmente progressivement la pondération, l'algorithme aligne de plus en plus les occurrences des motifs. Le nombre de motifs non-alignés re rapproche de 0.

### 4.4.1 Cas où un seul motif est commun aux deux séquences

Lorsqu'un seul motif est commun aux deux séquences, même s'il y a plusieurs occurrences de ce motif, le comportement de l'algorithme est simple : il n'y a pas de concurrence entre motifs. Si le motif n'est pas aligné par la programmation dynamique classique, en augmentant le coefficient de pondération multiplicatif, on va favoriser l'alignement des réalisations du motif.

La figure 4.6 donne un exemple, où les réalisations du motif ne font pas partie de la zone de plus grande similarité. Lorsqu'on choisit la valeur 2 pour le coefficient, la zone de similarité produite par l'algorithme n'est plus la même : il ne retient que la zone de similarité contenant les deux occurrences du motif.

2. Cet indice correspond au rapport du nombre de réalisations non alignées du motif sur le nombre total d'alignements possibles.

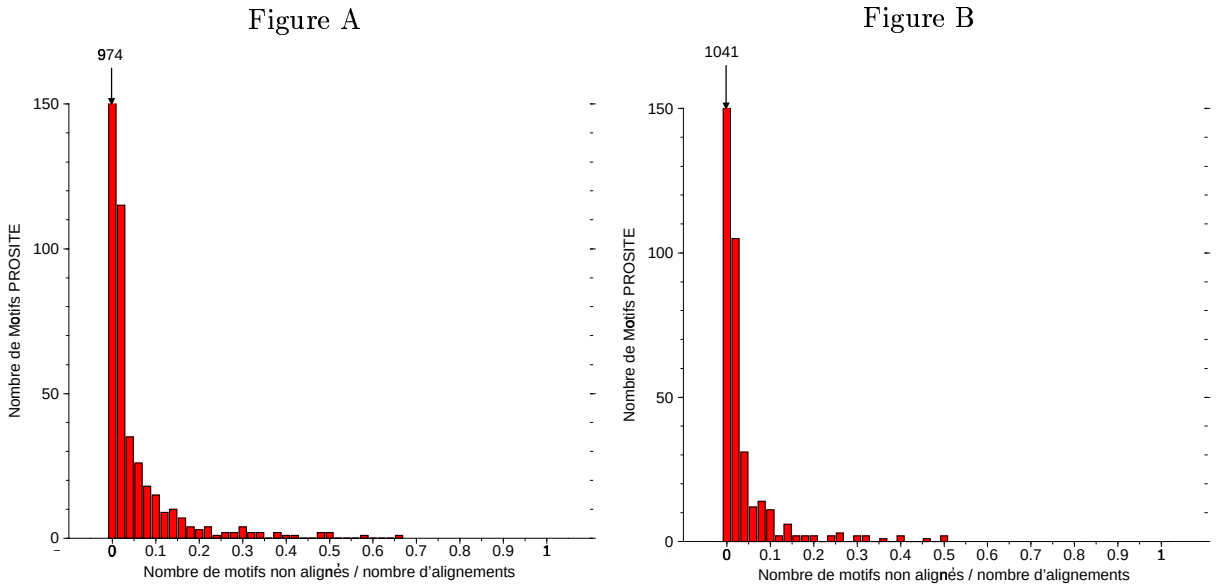



FIG. 4.5 - **Répartition des motifs de PROSITE Rel. 14.0 (vrais positifs)** *en fonction du nombre de motifs non-alignés par la programmation dynamique.*
**Figure A :** *Le coefficient vaut* 2.
**Figure B :** *Le coefficient prend la valeur* 3. *On peut comparer ces deux figures avec la figure 4.2.*

Lorsqu'on augmente le coefficient, on peut dans certains cas avoir un alignement avec les deux zones de similarité. La pondération contrebalance les insertions/délétions nécessaires au raccordement des deux zones de similarité. Dans l'exemple donné, ce n'est pas possible, puisque le coût de l'insertion/délétion est plus important que le score de la zone de similarité qu'on veut récupérer.

Il peut aussi y avoir concurrence entre plusieurs occurrences du même motif. Si on a deux zones de similarité contenant chacune une réalisation du motif, on va pouvoir en jouant sur le coefficient de pondération, aligner simultanément les deux zones de similarité. Le nombre maximum de motifs alignés est alors égal au minimum entre les deux nombres de réalisations du motif dans les séquences $A$ et $B$.

### 4.4.2 Cas où les deux séquences partagent plusieurs motifs

Comme nous venons de le voir, la programmation dynamique aligne dans beaucoup de cas, les motifs. Cependant dans les autres cas, on peut vouloir pondérer le motif, et par là, aligner des zones significatives (par présence de signatures d'activité biologique). Ces régions n'appartiennent pas forcement aux zones de hautes similarité.

Le premier exemple illustrera bien ce qui précède. Supposons qu'il existe deux zones de forte homologie très éloignées l'une de l'autre : une très longue insertion/délétion est nécessaire pour pouvoir visualiser dans le même alignement ces deux zones. La programmation dynamique classique ne trouve que l'une de ces deux zones. Par contre, s'il existe dans ces deux zones, un motif, l'algorithme SWM va permettre de raccorder ces deux zones dans le même alignement.

Choisissons un motif de la banque ***PROSITE*** dont le pourcentage d'occurrences alignées parmi les vrais positifs n'est pas proche de 1. Pour le motif $PS01288$, il y a 4 vrais positifs dans

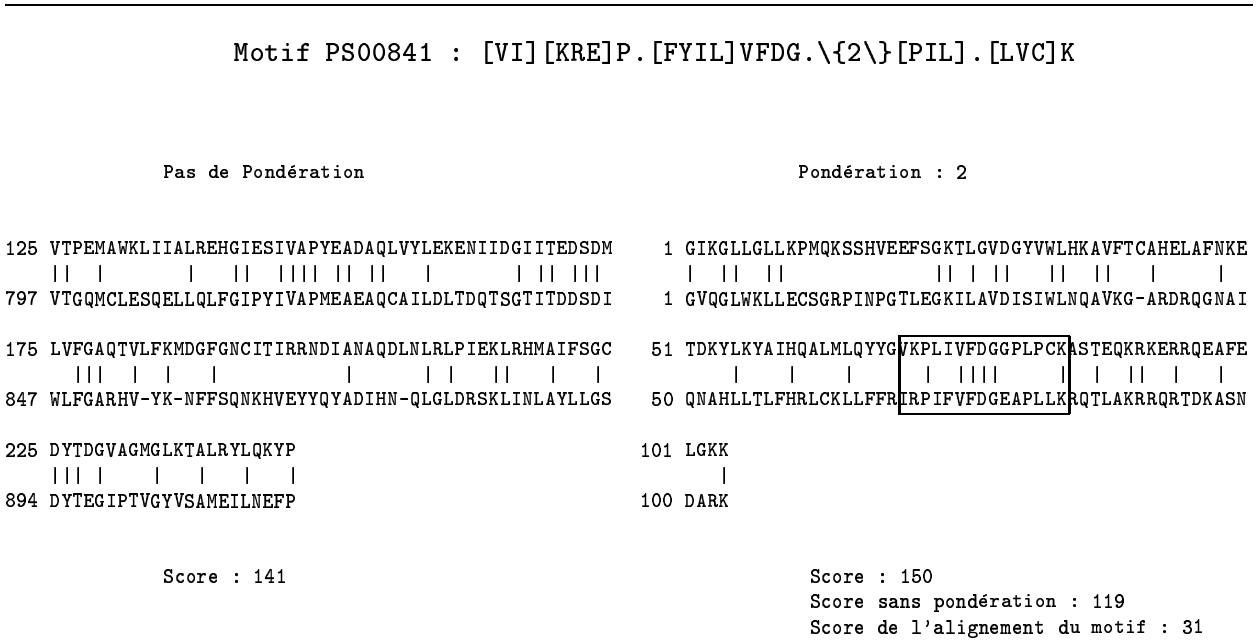



FIG. 4.6 - **Influence du coefficient sur l'alignement :** *Pour les séquences XPG_XENLA et EXO1_SCHPO, l'algorithme de SW n'aligne pas le motif PS00841 de PROSITE. Par contre, la pondération des occurrences de ce motif permet de mettre en évidence une autre zone de similarité, dont le score initial (*$119 = 150 - 31$*) est légèrement inférieur à celui trouvé par l'algorithme habituel. La partie encadrée correspond à la seule réalisation du motif PS00841.*

la banque SwissProt Rel. 35 : RTCB_ECOLI, Y682_METJA, YQ01_MYCTU et YT6J_CAEEL. On a 6 alignements possibles entre deux vrais positifs. On observe trois alignements qui ne mettent pas en correspondance les occurrences du motif. Le ratio du nombre de réalisations non alignées par le nombre d'alignements possibles concernant les vrais positifs, est de $3/6 = 0.5$. Les séquences RTCB_ECOLI et Y682_METJA possèdent ce motif, et la programmation dynamique simple ne fait pas apparaître l'alignement des occurrences du motif. Ces deux séquences partagent d'autres motifs :

| NOMS | Index PROSITE | Expressions régulières |
|---|---|---|
| PKC_PHOSPHO_SITE | PS00005 | [ST]-x-[RK] |
| CK2_PHOSPHO_SITE | PS00006 | [ST]-x(2)-[DE] |
| MYRISTYL | PS00008 | G-EDRKHPFYW-x(2)-[STAGCN]-P |
| UPF0027 | PS01288 | Q-[LIVM]-x-N-x-A-x-[LIVM]-P-x-I-x(6)-[LIVM]-P-D-x-H-x-G-x-G-x(2)-[IV]-G |

L'alignement sans pondération n'aligne pas le motif UPF0027, puisque la zone de haute similarité est très éloignée des deux occurrences de ce motif. Cette zone contient les deux motifs PS00006 et PS00008. L'alignement est donné dans la figure 4.7.

Cet alignement montre qu'il y a deux zones de similarité espacées par une très longue insertion/délétion. La pondération a été suffisante pour contrebalancer la pénalité qu'engendre cette très longue insertion/délétion [3]. Ce résultat est positif, puisqu'on a réussi à raccrocher deux zones de similarité espacées par une très longue insertion/délétion.

Si on regarde les alignements qu'on obtient pour différents coefficients de pondération, on constate que plus la pondération est importante, plus le nombre de motifs alignés est grand (Cf. figure 4.8). Il paraît indispensable de connaître le nombre maximal possible de réalisations de motifs alignées. D'après la figure 4.8, on n'arrive pas à aligner plus de 12 motifs. Ce nombre n'est pas nécessairement la limite supérieure, mais simplement dû à des pondérations mal adaptées.

### Comment calculer le nombre maximum de motifs alignés ?

En reprenant le cas précédent, on a plusieurs réalisations des différents motifs sur les deux séquences :

| | RTCB_ECOLI | Y682_METJA | Symbole |
|---|---|---|---|
| PKC_PHOSPHO_SITE | 8 | 11 | P |
| CK2_PHOSPHO_SITE | 2 | 6 | C |
| MYRISTYL | 11 | 13 | M |
| UPF0027 | 1 | 1 | V |
| Total | 22 | 31 | |

Certaines de ces réalisations se chevauchent, et l'algorithme SWM est incapable d'aligner simultanément de telles réalisations. D'autre part, l'ordre des réalisations des différents motifs a une grande importance. Si deux motifs apparaissent dans un ordre différent dans les deux séquences, il est impossible d'aligner simultanément les deux motifs.

3. Il est à remarquer qu'on pondère tous les motifs de la même manière, alors qu'on pourrait envisager une pénalisation différente pour chacun des motifs PROSITE.

| | Coefficients |
|---|---|
| Au Dessous | 1.0 |
| A Droite | 8.0 |

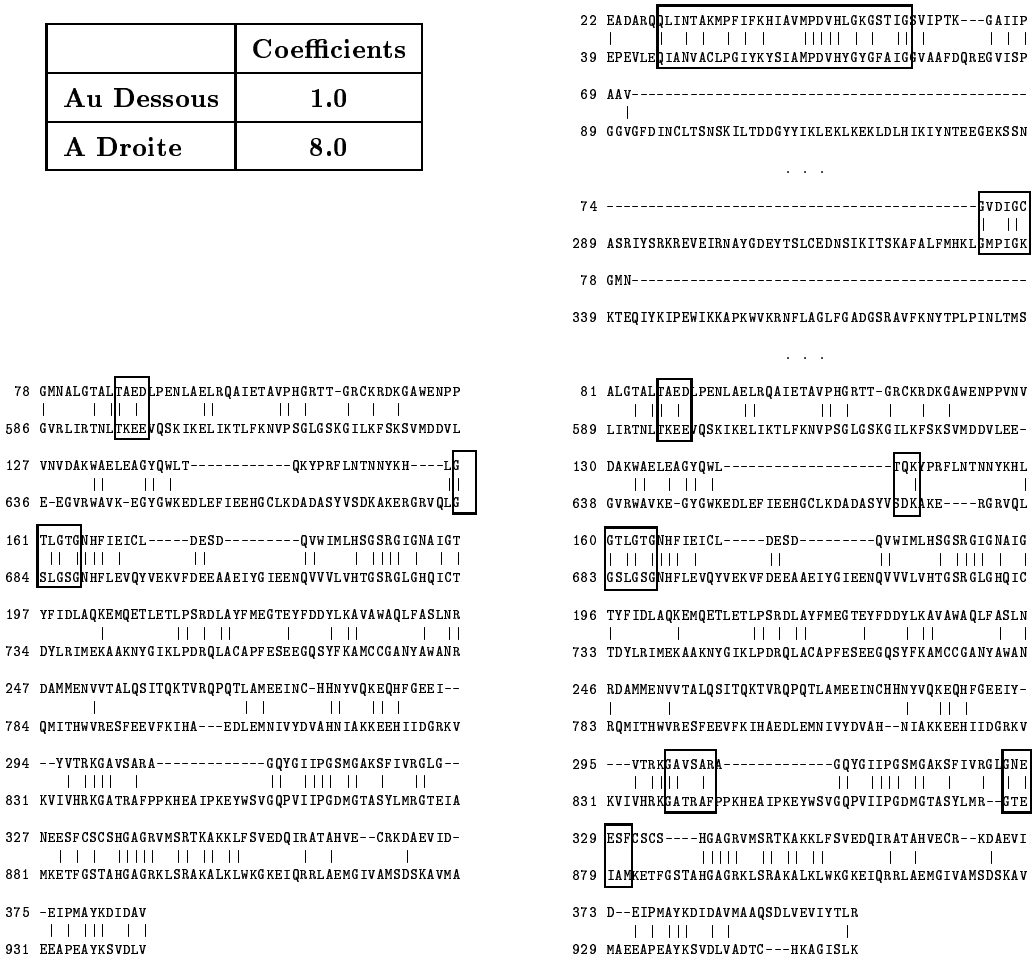


FIG. 4.7 - **Influence du coefficient sur l'alignement :** *En augmentant le coefficient, on permet à l'algorithme d'introduire de très longs gaps. Ainsi, pour les deux séquences RTCB_ECOLI et Y682_METJA, on fait apparaître deux zones de similarités. Les parties encadrées correspondent aux réalisations des différents motifs.*
*L'algorithme a réussi, grâce à la pondération des motifs, à raccrocher deux zones de similarité espacées par une très longue insertion/délétion.*

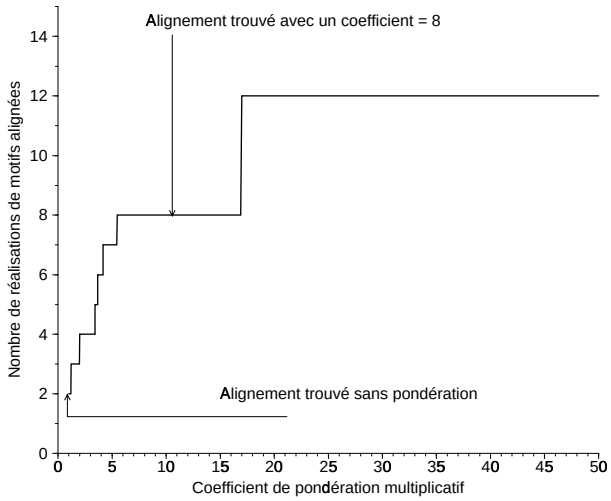



FIG. 4.8 - **Influence du coefficient sur le nombre de réalisations de motifs alignées :** *En augmentant le coefficient, le nombre de motifs alignés augmente. On arrive à saturation pour 12 motifs alignés.*

Pour connaître le nombre maximum de motifs alignés, on peut procéder comme suit :

1. On crée un alphabet de taille égale au nombre de motifs. Dans l'exemple précédent, l'alphabet sera $\mathcal{A} = \{P, C, M, V\}$ (Cf. tableau précédent) soit une lettre par motif.

2. Pour chaque séquence, on crée toutes les suites possibles de lettres sur ce nouvel alphabet représentant l'ordre d'apparition de ces motifs. Il y en a plusieurs, puisque, lorsqu'il y a chevauchement, un seul motif peut être aligné. Pour la séquence Y682_METJA, on a les deux suites de motifs suivantes :

   | | | | | | | | | | | | | | | | | | | | | | | | | | | |
   |---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
   | V | P | P | M | P | C | P | P | M | M | C | P | P | M | P | P | M | C | C | M | P | M | M | M | C | M | M | P |
   | M | P | P | M | P | C | P | P | M | M | C | P | P | M | P | P | M | C | C | M | P | M | M | M | C | M | M | P |

   puisque les motifs $V$ et $M$ qui se trouvent en début de séquence, se chevauchent.

   On obtient deux petites banques de séquences représentant les ordres d'apparitions des réalisations des motifs, l'une relative à la première séquence, et l'autre à la seconde. La seconde banque a 12 séquences.

3. Pour chaque couple de séquences $(A_i, B_j)$, $A_i$ appartenant à la première banque, et $B_j$ appartenant à la seconde, on cherche la plus longue sous-séquence commune. Pour ce faire, on peut utiliser l'algorithme SW, avec les paramètres suivants :

   - La matrice de substitution est la matrice identité,
   - $gapo = 0$,
   - $gape = 0$.

4. Le maximum obtenu sur toutes les comparaisons possibles entre ces deux banques de séquences donne le nombre maximal de motifs alignés. Les résultats pour l'exemple précédent consistent en $24 = 12 \times 2$ alignements dont le score maximum est 15, réalisé par plusieurs alignements :

```
1 VPPMPCPPMMCPPMPPMCCMPMMMCMMP          3 PMPCPPMMCPPMPPMCCMPMMMCMMP
  |  | |||||  || | |  | |  |  |  S=15     || |||||  || ||      |||  |   S=15
1 V--M-CPPMM-PP-P-M--M-M--C--P          1 PM-CPPMM-PP-PP------MMC--P

1 VPPMPCPPMMCPPMPPMCCMPMMMCMMP          1 MPPMPCPPMMCPPMPPMCCMPMMMCMMP
  |  | |||||  || |     | |||  |  S=15     |  | |||||  || | |  | |  |  | S=15
1 V--M-CPPMM-PP-P-----P-MMC--P          1 M--M-CPPMM-PP-P-M--M-M--C--P

3 PMPCPPMMCPPMPPMCCMPMMMCMMP            1 MPPMPCPPMMCPPMPPMCCMPMMMCMMP
  || |||||  |  |     |||||  |    S=15     |  | |||||  || |     | |||  | S=15
1 PM-CPPMM-P--P-----PMMMC--P            1 M--M-CPPMM-PP-P-----P-MMC--P
```

Même en augmentant le coefficient de pondération, on n'arrive pas à aligner 15 réalisations de motifs. On aligne 12 motifs dans l'ordre donné par la séquence suivante :

```
VPPMPPPMMMCP
```

Les trois motifs manquants n'ont pas pu être alignés à cause du modèle multiplicatif choisi pour la pondération des motifs.

## 4.5 Le modèle multiplicatif n'est pas toujours adapté

Lorsque l'expression régulière est assez lâche, l'alignement de deux occurrences du même motif peut avoir un score inférieur à 0. Dans ce cas là, une plus forte pondération ne forcera jamais l'alignement de tels motifs. Donnons un exemple. Le motif ***PROSITE*** PS00011 est défini par l'expression régulière :

```
.{12}E.{3}E.C.{6}[DEN].[LIVMFY].{9}[FYW]
```

Le début du motif est défini par la présence de 12 lettres quelconques. Biologiquement, cela signifit que le motif *utile* «E.{3}E.C.{6}[DEN].[LIVMFY].{9}[FYW]» ne peut pas se trouver à moins de 12 lettres du début de séquence. Considérons les deux réalisations de ce motif dans les séquences OSTC_MACFA et FA10_BOVIN :

```
OSTC_MACFA : WLGAPAPYPDPLEPKREVCELNPDCDELADHIGFQEAY
FA10_BOVIN : FLEEVKQGNLERECLEEACSLEEAREVFEDAEQTDEFW
```

L'alignement de ces occurrences est :

```
      .{12}    E .{3} E.C  .{6} [E].[F]   .{9}   [W]
43 FLEEVKQGNLER E CLE  EAC SLEEAR E V F EDAEQTDEF  W
    |          |      | |  |            |     |
 4 WLGAPAPYPDPL E PKR  EVC ELNPDC D E L ADHIGFQEA  Y
      .{12}    E .{3} E.C  .{6} [E].[L]   .{9}   [Y]
```

Le score de cet alignement est $-1$. Ce score est la somme du score de l'alignement des deux préfixes («FLEEVKQGNLER» et «WLGAPAPYPDPL») et du score d'alignement du motif *utile*. Le premier vaut $-14$ alors que le second vaut $+13$. Ce sont les préfixes qui sont à l'origine du score négatif de l'alignement. Les 12 lettres qui constituent les préfixes n'apportent pas d'information au motif. Il faudrait envisager de modifier l'algorithme afin de ne pas prendre en compte les préfixes et suffixes de ce type.

Pour favoriser cet alignement, on peut envisager de donner un bonus additif, mais qui ne valorise pas plus l'alignement de deux occurrences semblables que celui de deux occurrences dissemblables. Ce nouvel algorithme est pratiquement le même que celui donné par l'équation 4.2 : il suffit de remplacer la multiplication par une addition.

## 4.6 Data-Bank scanning avec SWM

Revenons au problème initial qui consiste à trouver toutes les séquences connues d'une banque de données, qui sont similaires à une séquence inconnue. Les différents algorithmes donnent un score qui va permettre de classer les séquences de la plus similaire à la plus distante. Qu'en est-il de l'algorithme SWM? L'information apportée par la mise en correspondance des motifs, est-elle suffisante pour améliorer le classement des séquences de la banque?

Puisqu'on compare une séquence à une banque, et que l'algorithme SWM ne diffère de l'algorithme SW que sur les couples de séquences possédant au moins un motif en commun, il faut choisir une séquence *requête* qui contienne un maximum de motifs. Si cette séquence ne possédait pas de motif, le score SWM serait égal au score SW. Le tableau 4.2 donne la répartition des motifs PROSITE dans les séquences Swissprot.

| Nombres de motifs PROSITE ($n$) | Nombres de séquences de Swissprot Rel. 35 ayant $n$ motifs |
|---|---|
| 1 | 24176 |
| 2 | 7610 |
| 3 | 2605 |
| 4 | 563 |
| 5 | 173 |
| 6 | 140 |
| 7 | 38 |
| 8 | 4 |

TAB. 4.2 - **Répartition des motifs PROSITE dans les séquences Swissprot**

Nous devons choisir une séquence qui possède beaucoup de références ***PROSITE***. La séquence FA12_HUMAN partage 7 motifs et 1 «*Matrix*». Nous allons comparer cette séquence à toutes celles qui partagent au moins l'un des motifs présents dans FA12_HUMAN : PS00021, PS00022, PS00023, PS00134, PS00135, PS01186 et PS01253. Le tableau 4.3 donne, pour chaque motif de FA12_HUMAN, le nombre de séquences de SwissProt Rel. 35 qui partagent ce motif, ainsi que le nombre total d'occurrences de ce motif.

Les motifs PS00022 et PS01186 ont été retirés pour des raisons de compléxité, le nombre d'occurences de ces motifs étant trop élevé. De plus on n'a pas retenu le motif PS00134, parce qu'il n'était pas très informatif : [LIVM]-[ST]-A-[STAG]-H-C. On ne tiendra donc compte que des motifs : PS00021, PS00023, PS00135, et PS01253. La banque de données des séquences partageant au moins

| motifs PROSITE | Nombres de séquences contenant le motif | Nombres d'occurrences du motif |
|---|---|---|
| PS00021 | 44 | 135 |
| PS00022 | 321 | 1045 |
| PS00023 | 29 | 55 |
| PS00134 | 288 | 288 |
| PS00135 | 287 | 287 |
| PS01186 | 337 | 1071 |
| PS01253 | 21 | 69 |

TAB. 4.3 - **Nombres d'occurrences dans SwissProt Rel. 35 des motifs présents dans FA12_HUMAN.**

l'un de ces quatre motifs, contient 321 séquences. Pour toute autre séquence, ne contenant aucun de ces motifs, l'algorime SWM est strictement équivalent à SW, et la comparaison des algorithmes est inutile.

Le tableau 4.4 donne les 20 plus grands scores lors de la comparaison de cette séquence contre toutes celles qui partagent avec elle au moins un motif. On y a rajouté certains résultats montrant l'influence du coefficient de pondération sur certains alignements. Plusieurs coefficients ont été utilisés. On observe une stabilité globale de l'ordre d'apparition des séquences. Cependant certaines séquences se retrouvent classées nettement avant, lorsqu'on utilise un coefficient supérieur à un. C'est le cas des séquences FINC_HUMAN, FINC_BOVIN, FINC_RAT, et dans une moindre mesure EL3B_HUMAN.

Par exemple, la séquence FINC_HUMAN apparaît au rang 273 avec le coefficient égal à 1 et au rang 87 avec le coefficient égal à 3. Augmenter le coefficient a permis de raccorder deux zones de similarité (Cf. figure 4.9).

L'avantage de l'algorithme SWM ne réside pas uniquement dans l'ordre dans lequel les séquences sont classées, mais aussi dans l'alignement lui-même. Il permet dans une large mesure de corriger le manque de sensibilité de l'algorithme SW, en permettant la création de très longues insertions/délétions.

## 4.7 Le problème de la pondération

La récurrence principale de l'algorithme 4.2 utilise des coefficients pour favoriser l'alignement de motifs. Le $Bonus(Motif_m)$ doit dépendre du motif aligné, puisque l'information portée par un motif, dépend de sa longueur et de sa grammaire. Mais il reste difficile de donner des valeurs numériques pour chacun des motifs de la banque ***PROSITE***.

Il est naturel de penser que le *Bonus* doit être d'autant plus grand que le motif est informatif. La quantité d'information apportée par un motif est inversement proportionnelle à la probabilité d'apparition de ce motif dans une séquence aléatoire. La théorie de l'information attribue à un événement de probabilité $p$, l'information $-log_2(p)$ [91]. On est donc amené à estimer la loi d'apparition d'un motif dans une séquence aléatoire.

La probabilité d'apparition d'une réalisation d'une *expression régulière* dans une chaîne de caractères peut être estimée par le dénombrement des mots vérifiant l'*expression régulière*. On calcule pour chaque motif, le nombre de mots réalisant l'expression régulière, puis on le divise par la longueur maximale du motif. On obtient une première approximation de la probabilité d'observer un motif en une position déterminée.

| | Coefficient = 1 | | | Coefficient = 2 | | | Coefficient = 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | séquences | scores | (1) | séquences | scores | (1) | séquences | scores | (1) |
| 1 | FA12_CAVPO | 2507 | 4 | FA12_CAVPO | 3019 | 4 | FA12_CAVPO | 3531 | 4 |
| 2 | FA12_BOVIN | 2425 | 4 | FA12_BOVIN | 2935 | 4 | FA12_BOVIN | 3445 | 4 |
| 3 | HGFA_HUMAN | 1193 | 4 | HGFA_HUMAN | 1485 | 4 | HGFA_HUMAN | 1777 | 4 |
| 4 | UROT_HUMAN | 712 | 3 | URT2_DESRO | 852 | 3 | URT2_DESRO | 997 | 3 |
| 5 | URT1_DESRO | 708 | 2 | URT1_DESRO | 850 | 3 | URT1_DESRO | 993 | 3 |
| 6 | URT2_DESRO | 707 | 3 | UROT_HUMAN | 849 | 3 | UROT_HUMAN | 986 | 3 |
| 7 | UROT_RAT | 694 | 3 | UROT_RAT | 832 | 3 | UROT_RAT | 970 | 3 |
| 8 | UROT_BOVIN | 687 | 3 | UROT_BOVIN | 826 | 3 | UROT_BOVIN | 965 | 3 |
| 9 | UROT_MOUSE | 675 | 3 | UROT_MOUSE | 808 | 3 | UROT_MOUSE | 941 | 3 |
| 10 | URTB_DESRO | 666 | 2 | URTB_DESRO | 769 | 2 | URTB_DESRO | 872 | 2 |
| 11 | UROK_BOVIN | 663 | 2 | UROK_BOVIN | 763 | 2 | UROK_BOVIN | 863 | 2 |
| 12 | UROK_RAT | 656 | 2 | UROK_HUMAN | 754 | 2 | UROK_HUMAN | 854 | 2 |
| 13 | UROK_HUMAN | 654 | 2 | UROK_RAT | 750 | 2 | UROK_RAT | 844 | 2 |
| 14 | UROK_PIG | 639 | 2 | UROK_PIG | 739 | 2 | UROK_PIG | 839 | 2 |
| 15 | UROK_PAPCY | 628 | 2 | UROK_PAPCY | 728 | 2 | UROK_PAPCY | 828 | 2 |
| 16 | UROK_MOUSE | 623 | 2 | UROK_MOUSE | 718 | 2 | UROK_MOUSE | 813 | 2 |
| 17 | PLMN_PIG | 587 | 2 | PLMN_PIG | 687 | 2 | PLMN_PIG | 787 | 2 |
| 18 | PLMN_BOVIN | 582 | 2 | PLMN_BOVIN | 682 | 2 | PLMN_BOVIN | 782 | 2 |
| 19 | PLMN_HUMAN | 582 | 2 | PLMN_HUMAN | 682 | 2 | PLMN_HUMAN | 782 | 2 |
| 20 | URTG_DESRO | 571 | 2 | URTG_DESRO | 674 | 2 | URTG_DESRO | 777 | 2 |
| | | | | | | | | | |
| 87 | CFAD_RAT | 347 | 1 | ENTK_MOUSE | 412 | 1 | **FINC_HUMAN** | 475 | 2 |
| 88 | TRY2_ANOGA | 345 | 1 | TRY2_ANOGA | 412 | 1 | **FINC_RAT** | 474 | 2 |
| 100 | EL2_BOVIN | 334 | 1 | TRYA_HUMAN | 397 | 1 | ***FINC_BOVIN*** | 462 | 2 |
| 218 | COGS_UCAPU | 247 | 1 | **FINC_HUMAN** | 304 | 2 | CTRB_GADMO | 379 | 1 |
| 219 | SNAK_DROME | 247 | 1 | **FINC_RAT** | 301 | 2 | TRYZ_DROME | 377 | 1 |
| 224 | NKP1_RAT | 237 | 1 | ***FINC_BOVIN*** | 295 | 2 | EL3B_HUMAN | 372 | 1 |
| 273 | **FINC_HUMAN** | 147 | 1 | TRYP_CHOFU | 243 | 1 | GRAB_MOUSE | 305 | 1 |
| 274 | ***FINC_BOVIN*** | 147 | 1 | MPRI_MOUSE | 240 | 1 | MCP1_PAPHA | 303 | 1 |
| 279 | **FINC_RAT** | 142 | 1 | GRL2_RAT | 229 | 1 | MCP2_MERUN | 285 | 1 |

TAB. 4.4 - **Comparaison SWM séquence contre banque : Extrait.** *On compare la séquence FA12_HUMAN à toutes les séquences de SwissProt Rel. 35 qui partagent l'un des motifs PROSITE suivants : PS00021, PS00023, PS00135 et PS01253. Les colonnes (1) donnent le nombre de motifs alignés.*

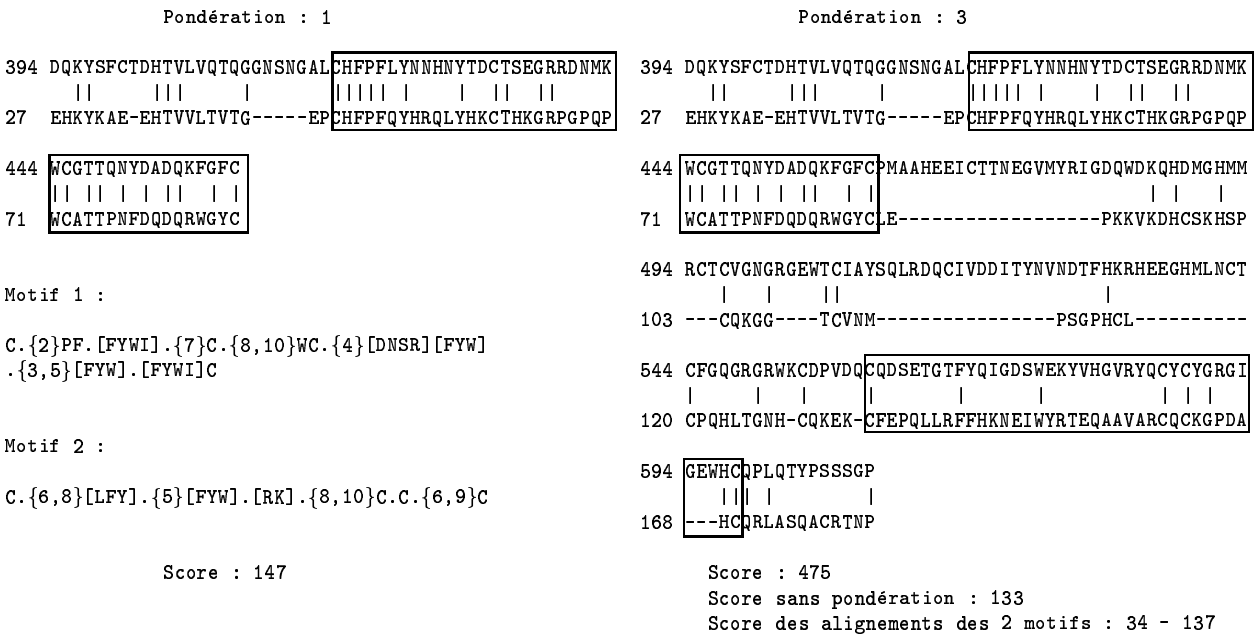



FIG. 4.9 - **Alignements des séquences FA12_HUMAN et FINC_HUMAN avec deux pondérations différentes.**

Cependant, l'apparition d'un motif dans une chaîne de caractères plus longue que le motif, dépend aussi de la longueur de la chaîne. On se retrouve face à un problème comparable à celui de la signification d'un alignement local.

### Utilisation de la méthode de Chen-Stein

Le comptage des mots dans une suite de lettres a été largement étudié. Et, un mot est un cas de motif dégénéré. Dans le cas de suites de lettres iid, la méthode de Chein-Stein a permis quelques avancées [28, 29] [44] [49, 50] [65]. L'approximation du nombre de «clumps» par un processus de Poisson est donnée par la méthode de Chen-Stein [27][110, 7] [15].

Il semble être possible d'utiliser la méthode de Chen-Stein pour donner une approximation du nombre d'occurrences d'un motif dans une chaîne de caractères très longue.

# Chapitre 5

# Interprétation statistique du score

*Lorsqu'une nouvelle protéine est séquencée, elle est tout de suite comparée à toutes celles connues, afin de déterminer les séquences connues qui ressemblent le plus à la séquence requête. Le biologiste est donc appelé à classer les alignements de telle manière que les séquences les plus proches de la séquence requête apparaissent en premier.*

*Mais les banques actuelles sont biaisées, puisque le séquençage n'a pas été exhaustif, mais guidé par différents intérêts (maladies génétiques, facilité de séquençage ...). Les résultats obtenus par un databank scanning ne sont par là même pas représentatifs d'un alignement de la séquence requête contre une séquence prise au hasard parmi toutes les séquences existantes. Comment savoir alors qu'un score d'alignement est significatif, si les seuls résultats que l'on connaisse sont biaisés ? On est tenté de corriger ce biais en essayant de modéliser des séquences aléatoires, représentatives de l'ensemble des séquences réelles, puis de déterminer une loi des scores sur ces séquences aléatoires.*

*Bien que les distributions des scores de certains algorithmes de recherche de similarités puissent être approximées, pour le cas de l'algorithme de Smith & Waterman, aucun élément théorique n'est accessible, sauf dans le cas dégénéré où les insertions/délétions sont interdites (la pénalité d'ouverture des insertions/délétions peut valoir l'infini).*

*Pour obtenir une signification statistique d'un score d'alignement Smith & Waterman, on construit la distribution empirique des scores lors d'un databank scanning, puis on estime la signification de chacun des alignements par rapport à cette distribution. Cette procédure est valide tant que la* séquence requête *est indépendante de la grande majorité des séquences de la banque de données et que ces séquences sont indépendantes entre elles. Dans ce cas, presque tous les scores obtenus peuvent être traités comme des variables indépendantes. On peut bien entendu faire d'autres hypothèses sur l'ensemble des séquences, ce qui mène à d'autres conclusions quant à la signification d'un score.*

*Pour modéliser le comportement aléatoire du score pour des séquences aléatoires, on peut envisager deux possibilités :*

- *dans le cas d'un* databank scanning *on cherche une distribution unique pour tous les scores [31]. Ceci équivaut à traiter l'ensemble des séquences comme un mélange de distributions de séquences de longueur et de composition en acides aminés différentes,*

- *on peut rechercher une distribution de scores pour chaque comparaison, ce qui implique que chaque score est replacé dans un modèle différent de séquences aléatoires, ayant chacune des longueurs et compositions en acides aminés propres, issues de celles des séquences initiales.*

*Les deux approches ne donnent pas la même signification statistique. Le choix de l'un des deux*

*modèles n'est pas sans conséquence. La seconde possibilité a l'avantage de prendre en compte les longueurs et compositions des séquences et de diminuer le biais qui en résulte.*

*D'autres travaux ont essayé de prendre en compte la dépendance en longueur des scores SW[51] [102][133], ou même de faire dépendre la loi du score de la longueur et de la composition en acides aminés des séquences [98].*

*Nous rendons compte ici d'une étude du comportement du Z-score, simple normalisation du score par rapport à des scores obtenus sur des séquences de mêmes longueurs et compositions en acides aminés que les séquences initiales.*

## 5.1 Significativité statistique du score Smith & Waterman

Le problème de la signification d'un score quel que soit l'algorithme sous-jacent, est devenu un axe de recherche important et indispensable à la classification des séquences. William Pearson déclarait en 1996 :

> « *The major technical improvement in protein sequence comparison over the past five years has been the incorporation of statistical estimates in widely used similarity searching programs.* »[106]

Dans le cas de l'algorithme sans insertion/délétion un modèle théorique existe (section 1.3.1). La rapidité de l'algorithme BLAST ainsi que son modèle probabiliste sous-jacent justifient l'utilisation intensive d'un tel algorithme malgré son manque de sensibilité. Pour l'algorithme de Smith & Waterman, aucun modèle probabiliste n'a été exhibé. Même si certains éléments du comportement ont été bien décrits [1], la loi du score Smith & Waterman reste inaccessible. Pour pallier cette carence, plusieurs approches ont été proposées.

La première approche pour résoudre ce problème a été la simulation. Lipman et *al.* [88] ont proposé d'utiliser un processus de Monte-Carlo pour estimer la signification statistique du score. La méthode utilisée est la comparaison du score initial avec les scores obtenus par comparaison de l'une des deux séquences avec des permutations de la deuxième séquence. Le **Z-score** est alors défini comme étant le score centré réduit. Le fait de replacer le score obtenu dans un échantillon

1. Arratia et Waterman ont pu démontrer qu'il existait deux types de comportement asymptotique pour le score NW et SW.

**Théorème 5.1.1** *(Arratia et Waterman, 1994, [8]) Soient* $A = A_1, A_2, \ldots$ *et* $B = B_1, B_2, \ldots$ *deux suites de lettres iid. Considérons le score de l'alignement optimal* $SW_n = SW(A[1,n], B[1,n])$ *avec une matrice de substitution symétrique, et une fonction de pénalisation des gaps subadditive :* $g(l+k) \leq g(l) + g(k)$.

*Posons* $a = lim_{k\to\infty} \frac{E(SW_k)}{k}$.

*Si* $a > 0$, *alors* $SW_n$ *a une croissance linéaire de coefficient* $a$.

$$\frac{SW_n}{n} \underset{n\to\infty}{\longrightarrow} a \text{ en probabilité.}$$

*Si* $a < 0$, *alors* $SW_n$ *a une croissance logarithmique avec les coefficients* $[b, 2b]$ *où* $b$ *est défini par*

$$b = \max_{q\geq 0} \frac{q}{lim\{-\frac{1}{k} log P(SW_k \geq qk)\}}$$

$$P\left\{(1-\varepsilon)b < \frac{SW_n}{\log n} < (2+\varepsilon)b\right\} \underset{n\to\infty}{\longrightarrow} 1$$

de scores aléatoires, en tenant compte des compositions en acides aminés et des longueurs, permet d'atténuer le biais dû à des compositions différentes et celui dû aux longueurs.

Vingron & Waterman[133, 128] ont développé une méthode d'approximation poissonnienne pour estimer la significativité d'un score SW. Les auteurs utilisent l'heuristique "*Poisson clumping*" pour décrire le comportement des scores. Plaçons-nous dans le cas où la moyenne des éléments de la matrice de substitution est négative. Les alignements de scores positifs sont alors des événements rares. Notons de plus que les scores positifs apparaîssent en groupes. Si on observe une grande valeur en position $(i, j)$, alors toutes les valeurs voisines seront grandes (Cf. section 2.2, p. 58). Tous les alignements associés à ces valeurs ont en commun au moins une substitution. Ils constituent un *clump*, c'est-à-dire un ensemble d'alignements optimaux/sous-optimaux chevauchants.

Le modèle de Vingron & Waterman repose sur l'heuristique d'Aldous [2]. Les *clumps* sont modélisés par un processus de Poisson, et on assigne à chaque *clump* une taille indépendante. Le nombre de clumps dont le score est supérieur à $t$, a une distribution de Poisson de paramètres $\lambda_{m,n}(t)$. Les auteurs posent $\lambda_{m,n}(t) = \gamma mnp^t$ où $m$, $n$ sont les longueurs des séquences.

La probabilité d'obtenir un score plus petit ou égal à $t$ est alors $exp(-\gamma mnp^t)$, où $m$, $n$ sont les longueurs des séquences. Les paramètres $(\gamma, p)$ sont estimés sur des scores aléatoires obtenus avec des séquences aléatoires générées à partir d'une distribution donnée des acides aminés. Ces paramètres dépendent de la distribution en acides aminés choisie, et des paramètres de l'algorithme. Deux estimations du couple $(\gamma, p)$ sont possibles.

- L'estimation directe consiste à calculer un grand nombre de scores SW, à construire la distribution empirique, et à faire une regression linéaire sur les données après transformation.

- L'estimation par *declumping* nécessite le calcul des scores optimaux et sous-optimaux (Algorithme de Waterman & Eggert [132], section 1.5.2, p. 41). L'idée de base est d'interpréter la moyenne du processus de Poisson $\lambda_{m,n}(t)$ comme le nombre moyen de *clumps* dont le score dépasse $t$. En calculant pour differentes valeurs de $t$ la moyenne du processus de Poisson puis en représentant les résultats en échelle logarithmique, on obtient une ligne droite. L'estimation des paramètres est alors immédiate.

On appellera *p-value* la probabilité d'obtenir un score au moins égal à une valeur donnée $t$. Pour le modèle de Vingron & Waterman, la *p-value* est donnée par: $1 - exp(-\gamma mnp^t)$. La significativité d'un alignement est donnée par la *p-value*, c'est-à-dire la probabilité d'obtenir un score au moins égal à celui obtenu.

Deux points sont à relever.

1. La *p-value* donnée par leur approche n'est valable que pour la comparaison d'une séquence contre une banque. Certes les paramètres $p$ et $\gamma$ ne sont calculés qu'une fois. Ces deux paramètres dépendent de la composition en acides aminés, choisie pour la génération des séquences aléatoires. Que l'on choisisse un modèle de séquences aléatoires du type **M0** (lettres indépendantes identiquement distribuées) ou un modèle **M1** (modèle de Markov d'ordre 1), les paramètres du modèle sont estimés sur la banque de séquences. Lorsqu'on change de banque de données, le modèle de séquences aléatoires change et les paramètres $(\gamma, p)$ doivent être réestimés.

   De plus $\gamma$ est corrigé par Estimation du Maximum de Vraissemblance. $\gamma$ devient dépendant de la banque de données et de la séquence requête.

   Cette méthode permet de classer par ordre de *p-value* les differentes séquences d'une banque de données qui partagent une zone similaire avec la séquence requête. Cependant elle ne

permet pas de comparer des alignements entre eux s'ils n'ont aucune séquence commune, puisque le modèle dépend de la séquence requête.

2. Le score dépend de la ressemblance des compositions en acides aminés des deux séquences. Si deux séquences ont des distributions très proches, elles auront d'autant plus de chance d'avoir un score élevé. La signification statistique de l'alignement doit tenir compte de cette ressemblance, et donc des *deux* compositions en acides aminés.

   La méthode d'approximation poissonnienne ne prend pas en compte les compositions en acides aminés des deux séquences, mais uniquement une composition fixe (les séquences aléatoires sont générées à partir de cette distribution), souvent choisie comme étant celle de la banque. Un manque de sélectivité peut alors apparaître.

   Un score élevé obtenu avec deux séquences de la même base de données, ayant deux compositions en acides aminés similaires, mais distinctes de la distribution globale de la base de donneés, aura une très faible probabilité alors que ce score n'est pas forcément significatif.

   **Exemple :** Les séquences $YBA4\_YEAST$ et $YJK9\_YEAST$ tirées de **SWISSPROT** ont des compositions similaires, avec chacune une très forte proportion de $S$. Par défaut, l'algorithme de Vingron et Waterman prend en compte une composition fixe, celle de McCalden-Argos, qui ne présente pas la spécificité des deux séquences, à savoir avoir beaucoup de $I$, $L$, $S$ et peu de $A$,$G$ (Cf. figure 5.1-A).

   Pour ces séquences particulières, la *P-value* donnée par la méthode poissonnienne de Waterman et Vingron, n'est pas représentative :

```
P-value parameters:     gamma = 0.0192192,
                        p     = 0.938912,
P(Score[1] > 53) ~ 2.62658e-10
```

   Le fait d'imposer la composition en acides aminés pour l'évaluation de la significativité du score entraîne un biais lorsque les séquences à comparer ont des compositions similaires mais distantes de celle pour le génération de séquences aléatoires. La position du score obtenu, 53, dans un échantillon de scores obtenus par le modèle de Vingron & Waterman, donne une significativité très importante. Si on replace ce score (53) dans un échantillon de scores obtenus sur des séquences ayant les mêmes compositions en acides aminés, la significativité est beaucoup plus faible (Cf. figure 5.1-B).

## 5.2 Le Z-score

Face à ces remarques, nous avons choisi de nous focaliser sur la méthode traditionnellement utilisée pour le calcul de la significativité statistique : les simulations. La méthode utilisée est la comparaison du score initial avec les scores obtenus par comparaison de l'une des deux séquences avec des séquences aléatoires. La séquence aléatoire est ici une **permutation** de la séquence initiale. Elle partage la même distribution en acides aminés et a la même longueur que la séquence initiale. Ce processus prend en compte le biais dû aux compositions différentes en acides aminés. Cette méthode est largement utilisée, on peut citer pour mémoire, que le Z-score est implémenté dans le programme RSS [105] ainsi que dans Bestfit [35].

Le score ne suit pas une distribution normale ni même une autre distribution connue valable pour toutes les séquences. La loi des scores dépend au moins des longueurs des deux séquences. On

Figure A

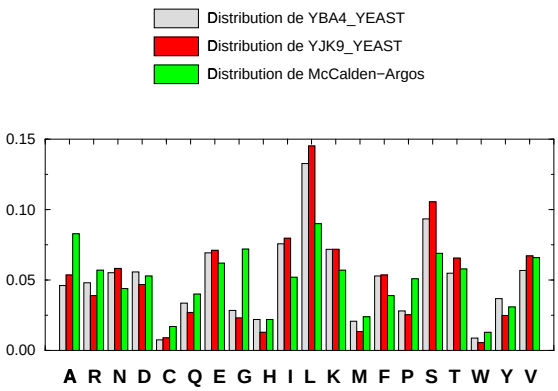



Figure B

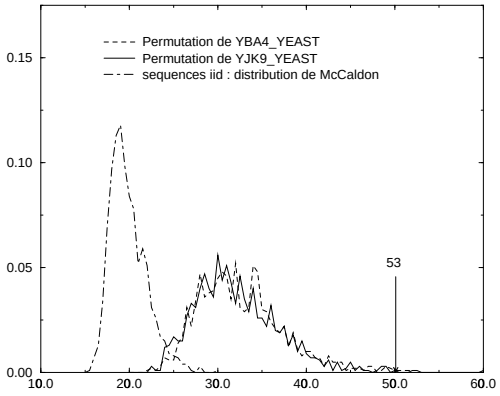



FIG. 5.1 - **Comparaison des séquences YBA4_YEAST et YJK9_YEAST.**
**A :** *Composition en acides aminés des séquences YBA4_YEAST et YJK9_YEAST.*
**B :** *Dépendance de la significativité du score en fonction du choix des séquences aléatoires. Si on normalise le score SW (ici 53), par rapport à un échantillon de scores obtenus sur des séquences aléatoires iid, de composition fixe, mais différente de celles des séquences, on risque de surévaluer la significativité.*

ne peut donc pas calculer de *P-value*, c'est-à-dire la probabilité d'obtenir un score au moins égal à celui obtenu. Le Z-score ne représente pas une probabilité mais une normalisation du score par rapport à un ensemble de valeurs aléatoires :

$$\text{Z-score} = \frac{\text{score - moyenne des scores aléatoires}}{\text{écart-type}} \tag{5.1}$$

Soit $A$ et $B$ deux séquences de longueurs respectives $l_A$ et $l_B$. Appelons $\chi$, le processus de permutation d'une séquence : $\chi(séquence) = séquence\ permutée$.
$E_\chi[S(\chi(A), \chi(B))]$ est l'espérance mathématique du score $S(\chi(A), \chi(B))$ pour le processus $\chi$. Pour le calcul du Z-score, il faudrait permuter les deux séquences, le Z-score serait alors donné par :

$$Z(A,B) = \frac{S(A,B) - E_\chi\Big[S(\chi(A),\chi(B))\Big]}{\sqrt{E_\chi\Big(\Big[S(\chi(A),\chi(B)) - E_\chi\Big[S(\chi(A),\chi(B))\Big]\Big]^2\Big)}} \tag{5.2}$$

Plusieurs processus sont possibles pour permuter une séquence. On peut se référer à [88] où ce point est discuté dans le cas des séquences nucléiques. Le processus le plus répandu préserve la composition globale en acides aminés. On peut toutefois envisager de garder la composition *locale* en acides aminés en faisant une permutation locale, ou de garder la composition en diacide. On choisit pour toute la suite de cette étude le processus de permutation le plus répandu.

Pour ce processus le nombre de séquences à permuter est si grand que la formule (5.2) n'est pas envisageable. On introduit une première simplification, qui consiste à ne permuter qu'une seule des

deux séquences. L'expression du Z-score devient :

$$Z(A,B) = \frac{S(A,B) - E_\chi\Big[S(A,\chi(B))\Big]}{\sqrt{E_\chi\Big(\Big[S(A,\chi(B)) - E_\chi\Big[S(A,\chi(B))\Big]\Big]^2\Big)}} \tag{5.3}$$

Cependant, le calcul de la moyenne $m = E_\chi[S(A,\chi(B))]$ et de l'écart-type

$$\sigma = \sqrt{E_\chi[(S(A,\chi(B)) - m)^2]}$$

reste hors de portée à cause du nombre important de permutations possibles.

### 5.2.1 Nombre de permutations

Le nombre de séquences qu'on peut obtenir par permutation est fini. Supposons que la taille de l'alphabet $\mathcal{A}$ est $N$ : $\mathcal{A} = \{A_1, A_2, \ldots A_N\}$. Combien de séquences peut-on obtenir en permutant une séquence de longueur $L$ et de composition $(n_i)_{i=1,\ldots,N}$ ? $n_i$ est le nombre de fois que la lettre $A_i$ apparaît dans la séquence. On a $\sum_{i=1}^{N} n_i = L$.

Pour résoudre ce problème de dénombrement,

- on doit d'abord choisir $n_1$ places dans la séquence à créer, où apparaîtra la première lettre de l'alphabet $A_1$, c'est-à-dire choisir $n_1$ parmi $L$ sans ordre :
  il y a $\dfrac{L(L-1)(L-2)\ldots(L-n_1+1)}{n_1!}$ possibilités.
- Ensuite, on place les $n_2$ lettres $A_2$ parmi les $L - n_1$ places restantes.
  Il y a $\dfrac{(L-n_1)(L-n_1-1)(L-n_1-2)\ldots(L-n_1-n_2+1)}{n_2!}$ possibilités.
- On procède de la même manière pour les lettres suivantes, $(A_i)_{i=3,\ldots,N}$. Posons $N_{i-1} = \sum_{i=1}^{n_i-1} n_i$. Pour choisir la place des $n_i$ lettres $A_i$ ,
  il y a $\dfrac{(L-N_{i-1})(L-N_{i-1}-1)(L-N_{i-1}-2)\ldots(L-N_{i-1}-n_i+1)}{n_i!}$ possibilités.

Il existe un autre raisonnement. Pour une séquence de longueur $L$ ne contenant pas deux fois la même lettre, on peut permuter la séquence de $L!$ façons différentes. Mais, lorsqu'une même lettre apparaît au moins deux fois, on compte plusieurs fois la même séquence. Pour une séquence donnée, il y a $\prod_{i=1}^{N}(n_i!)$ façons de la permuter et de retomber sur la même séquence. Globalement, il y a $\frac{L!}{\prod_{i=1}^{N}(n_i!)}$ séquences différentes de même composition.

Puisqu'il s'agit d'un coefficient du binôme généralisé[2], le nombre minimum de permutations est atteint pour une composition non homogène en particulier pour $n_i = L$ et $n_j = 0, \forall j \neq i$. Le nombre maximum de permutations est atteint pour une composition homogène, c'est-à-dire pour $n_1 = n_2 = \ldots = n_N = \frac{L}{N}$.

> **Exemple :** si la longueur de la séquence est $L = 100$, avec exactement 5 résidus pour chaque acide aminé, le nombre total de permutations est de l'ordre de $2.4 \times 10^{114}$.

2. La loi binomiale généralisée est appelée la loi multinomiale.

Etant donné le nombre important de permutations, on utilise un processus de Monte-Carlo pour le calcul du Z-score. Pour un couple de séquences $A$ et $B$, le score SW est comparé aux scores SW entre $A$ et un certain nombre $N$ de séquences permutées de $B$. Sur les $N$ scores aléatoires, on détermine la moyenne $\tilde{m}$ et l'écart-type $\tilde{\sigma}$. Le Z-score est défini par :

$$Z(A,B) = \left(\frac{S(A,B) - \tilde{m}}{\tilde{\sigma}}\right) \tag{5.4}$$

### 5.2.2 Asymétrie du Z-score

Dans la définition du Z-score, l'une des deux séquences uniquement est permutée. Permuter la deuxième séquence mène à une valeur du Z-score $Z_1(A,B)$ différente de celle $(Z_2(A,B))$ obtenue en permutant la première.

Dans la majorité des cas, le choix de la séquence à permuter n'influe pas sur le résultat du Z-score (Cf. figure 5.1-B). Cependant, dans le cas où l'une des deux séquences possède une composition en acides aminés vraiment différente de l'autre, le choix peut être crucial. Donnons un exemple. Soient les séquences suivantes :

```
>SR40_YEAST
MASKKIKVDEVPKLSVKEKEIEEKSSSSSSSSSSSSSSSSSSSSSSSSSG
ESSSSSSSSSSSSSSDSSDSSDSESSSSSSSSSSSSSSSSSDSESSSESDS
SSSGSSSSSSSSSDESSSESESEDETKKRARESDNEDAKETKKAKTEPES
SSSSESSSSGSSSSSESESGSESDSDSSSSSSSSSDSESDSESDSQSSSS
SSSSDSSSDSDSSSSDSSSDSDSSSSSSSSSSDSDSDSDSSSDSDSSGSS
DSSSSSDSSSDESTSSDSSDSDSDSDSGSSSELETKEATADESKAEETPA
SSNESTPSASSSSSANKLNIPAGTDEIKEGQRKHFSRVDRSKINFEAWEL
TDNTYKGAAGTWGEKANEKLGRVRGKDFTKNKNKMKRGSYRGGSITLESG
SYKFQD
```

```
>YNJ5_YEAST
MVHITLGQAIWVSVKPIIKIYLIIGVGFLMAKMGILTVEATRIISDIVLT
VLLPSLSFNKIVANIEDKDIKSVGIICLSALLIFGSGFFFAYVVRLFLPV
PKQWYGGILAGGMFPNISDLPIAYLQSMDQGLVFSEEEGNKGVANVIIFL
TMFLICIFNLGGFRLIESDFEYNDDESAVRVSETTKTQPAVSANTTNTDT
SERFFSNEQQLFNNKYTARDSLTEAIGTKGENADVPPISRRSTNSIAPLS
LPDTSSNSKITKPVQVKARNTIACTQSEESQATRGSNPLDSQSSASTIHS
YNTSESYESSIDTMRARRTASQPRAYNTTTLLEENCLDEKCPKNMSMAAL
EPIRSIDMRALPSQNIHHLIREYSNVDQYGHQRRNSSLRGADMNDVHSIS
SNSTLQTIKTANLTRILTSDATVSKKDIETSGESLPQWMRKFSLTPLLVF
FLKNCLRPCSMAVIIALTVAFIPWVKALFVTTANTPHISQAPDNAPPLSF
FMDFTGYVGAACVPFGLILLGATLGRLKIGNLYPGFWKAAVTLVILRQCV
MPIFGVLWCDRLVKAGWVNWQDDRMLLFVIAISWNLPTMTTLIYFTASFT
PPETTAPIQMECVSFFLMLQYPLMVVSLPFLVSYFLKVQMNL
```

La première séquence $SR40\_YEAST$ possède une grande proportion de $S$. Supposons que l'on choisisse de permuter cette séquence. Lors des permutations successives, les $S$ vont se trouver statistiquement encore côte à côte. L'alignement SW (ci-dessous) utilise fortement les suites consécutives de $S$. Les scores aléatoires vont aussi profiter des $S$ consécutifs. Ils seront donc du même ordre de grandeur que le score initial. Le Z-score en permutant $SR40\_YEAST$ sera faible.

**ALIGNEMENT INITIAL de SR40_YEAST avec YNJ5_YEAST**

```
168 SDFEYNDDESAVRVSETTKTQPAVSANTTNTDTSERFFSNEQQLFNNKYT
    || | +   |+   | ++ +    |+ + ++ +     |+     ++  +
 71 SDSESSSSSSSSSSSSSSSDSESSSESDSSSSGSSSSSSSSSDESSSES

218 ARDSLTEAIGTKGENADVPPISRRSTNSIAPLSLPDTSSNSKITKPVQVK
      +  |+    + +| |     +  |   +  |   +|| |  +   +
121 ESEDETKKRARESDNEDAKETKKAKTEPESSSSSESSSSGSSSSSESESG

268 ARN-TIACTQSEESQATRGSNPLDSQSSASTIHSYNTSESYESSIDT
    + + + + + |  |  +   +  |||||+|+  | ++|+|  || |+
171 SESDSDSSSSSSSSSDSESDSESDSQSSSSSSSSDSSSDSDSSSSDS

                    Score = 78
```

Par contre, si on permute la séquence YNJ5_YEAST, on va casser la structure qui forme l'alignement de score 78. Les motifs qui «*accrochent*» une suite de $S$, comme par exemple $DSQSSAST$,

sont détruits par les permutations. Les scores obtenus seront plus petits, et le Z-score plus grand (Cf. figure 5.3).

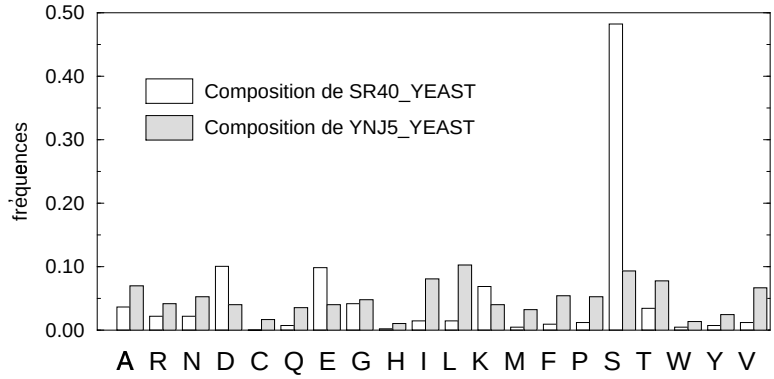



FIG. 5.2 - **Composition en acides aminés** *des séquences SR40_YEAST et YNJ5_YEAST.*

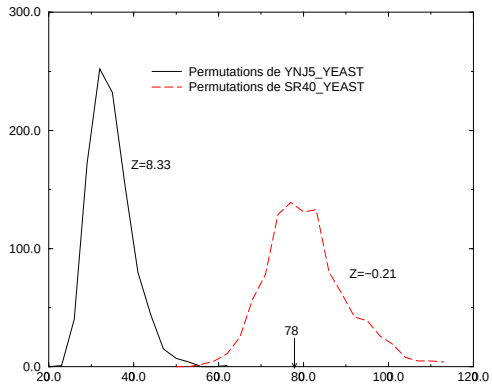



FIG. 5.3 - **Répartitions des scores** *après permutation de l'une des deux séquences. Le score initial entre les deux séquences est de 78 (Matrice Blosum62, go = 10 et ge = 1). La permutation de SR40_YEAST mène à un Z-score de $Z_1 = -0.21$ alors que la permutation de YNJ5_YEAST mène à un Z-score de $Z_2 = 8.33$.*

D'autres exemples peuvent être trouvés, où les deux Z-scores sont encore plus éloignés. Il s'agit encore de cas où l'une des deux séquences est de faible compléxité. Regardons par exemple les séquences YIR7_YEAST et SR40_YEAST. Lorsqu'on permute la séquence SR40_YEAST, le Z-score est de 3.37, alors que lorsqu'on permute YIR7_YEAST, le Z-score monte à 21.6 et l'alignement

semble satisfaisant (SW = 258 avec la matrice BLOSUM62, *gapo* = 10, *gape* = 1).

**ALIGNEMENT INITIAL de SR40_YEAST avec YIR7_YEAST**

```
  33 SSSSSSSSSSSSSSSSSGESSSSSSSSSSSSSSDSSDSSDSESSSSSSSS
     +   +++| |  +  || + |+||       |||  ||+     |+++|+
1081 AEKQATASMSIVALPSSFQESNSSDRYRKYCSSDE-DSNTCIHGSANAST

  83 SSSSSSSSDSESSSESDSSSSGSSSSSSSSSDESSSESESEDETKKRARE
     ++|+++ + + ++  ++++++ |+++++++|  +|+ + +   |
1130 NASTNAITTASTNVRTNATTNASTNATTNASTNASTNATTNASTNATTNS

 133 SDNEDAKETKKAKTEPESSSSSESSSSGSSSSSESESGSESDSDSSSSSS
     | |     |  | |   +|+++ +| +  +|++ +|| + | + +++ |+
1180 STNA----TTTASTNVRTSATTTASINVRTSATTTESTNSSTNATTTEST

 183 SSSDSESDSESDSQSSSSSSSSDSSSDSDSSSSDSSSDSDSSSSSSSSSS
     +|| + + +||   ++|++|++ ++| +  +|+ ++  ++||+|+++++|
1226 NSSTNATTTES---TNSNTSATTTASINVRTSATTTESTNSSTSATTTAS

 233 DSDSDSDSSSDSDSSGSSDSSSSSDSSSDESTSSDSSDSDSDSDSGSSSE
      +   | +++ | +| ++ +++ | +|+  +|+++|++| +++ +  |+
1273 INVRTSATTTKSINSSTNATTTESTNSNTNATTTESTNSSTNATTTESTN

 283 LETKEATADESKAEETPASSNESTPSASSSSSANKLNIPAGTDEIKEGQR
       |  ||  ||    | |++ ||| | +|+++    |  |  |  |+|
1323 SSTN-ATTTESTNSNTSAATTESTNSNTSATTTESTNASAKEDANKDGNA

 333 KHFSRVDRSKINFEAWE
     +       + || |+++
1372 EDNRFHPVTDINKESYK

                    Score = 258
```

Dans les deux exemples précédents, lorsqu'on permute l'une des deux séquences, la structure qui forme l'alignement n'est pas cassée (lorsqu'on permutte une suite de $S$, on obtient encore une suite de $S$). On peut d'ailleurs distinger les deux exemples :

- SR40_YEAST et YNJ5_ YEAST : l'alignement initial ne fait intervenir dans la séquence SR40_YEAST que la zone de faible complexité. On a un Z-score (en permutant SR40_YEAST) de $-0.21$.
- SR40_YEAST et YIR7_YEAST : l'alignement initial fait aussi intervenir la fin de la séquence SR40_YEAST, où la complexité est beaucoup plus forte. Le Z-score (en permutant SR40_YEAST) s'en trouve modifié : $Z = 3.37$.

Lorsqu'on permute l'autre séquence (YNJ5_YEAST ou YIR7_YEAST), la structure intervenant dans l'alignement est cassée. Pour estimer la moyenne et l'écart-type il est préférable de s'assurer que les structures ayant permis l'alignement initial ne sont pas conservées. On est amené à modifier la définition du Z-score : on calcule les deux Z-scores, en permutant l'une des deux séquences, puis en permutant l'autre.

$$\begin{aligned} Z_1(A,B) &= \frac{S(A,B) - \widetilde{m_1}}{\widetilde{\sigma_1}} \\ Z_2(A,B) &= \frac{S(A,B) - \widetilde{m_2}}{\widetilde{\sigma_2}} \end{aligned}$$

où $\widetilde{m_1}$ et $\widetilde{\sigma_1}$ sont la moyenne et de l'écart-type empiriques sur l'echantillon obtenu en permutant la séquence $A$, et $\widetilde{m_2}$ et $\widetilde{\sigma_2}$ sont la moyenne et de l'écart-type empiriques après avoir permuté la

séquence $B$. On redéfinit le Z-score par :

$$Z(A,B) = min\,(Z_1, Z_2) \tag{5.5}$$

Le choix de la fonction *min* provient de notre volonté d'éliminer au maximum les grands Z-scores correspondant à des alignements biologiquement non pertinents.

En reprennant les deux exemples précédents, l'une des deux séquences (SR40_YEAST) est de faible complexité : elle contient beaucoup de $S$. L'alignement initial profitant de ces $S$ successifs n'est donc pas biologiquement très significatif. Dans de tels cas, la fonction *min* permet de diminuer la significativité du score en fonction de la faible complexité des séquences.

### 5.2.3 Le score et le Z-score ne sont pas équivalents.

Le Z-score est nettement plus coûteux en temps puisqu'il nécessite $2 \times N$ comparaisons SW par paire de séquences. Ce calcul n'est utile que s'il permet de mettre en relief des similarités statistiquement significatives que le score ne souligne pas. Si le Z-score était proportionnel au score, ou s'il en résultait par une transformation simple, il ne présenterait aucun intérêt. Pour tester la non-équivalence entre ces deux indices de similarité, les comparaisons systématiques de toutes les séquences du génome de *Haemophilus Influenzae* ont été effectuées en utilisant à la fois le score et le Z-score. Ce génome comporte 1680 séquences. 1 410 360 scores et Z-scores ont été générés.

A chaque paire de séquences, on associe la probabilité sur ce génome d'avoir un score au moins égal à celui observé. De la même manière, on calcule la probabilité dans ce génome d'avoir un Z-score au moins égal à celui observé. On compare ces deux probabilités $P(S \geq s)$ et $P(Z \geq z)$. Le graphique 5.4 affiche dans le plan défini par ces deux probabilités toutes les paires de séquences. Si le Z-score et le score étaient équivalents, tous les points devraient être placés près de la diagonale.

On observe que pour les petites probabilités (inférieures à 0.0009), les Z-scores sont bien corrélés aux scores (Cf. figure 5.4, intersection des zones **A** et **B**). Dans cette partie du graphique, les scores et les Z-scores sont très grands : la valeur du score qui correspond à une probabilité de 0.0009 est $S = 25$, et le Z-score qui correspond à la même probabilité est $Z = 8.0$. Tous les alignements apparaissant dans cette zone ont un score $S \geq 25$ et un Z-score $Z \geq 8.0$.

Pour ces grands scores ($S \geq 25$), la similarité entre les deux séquences est très forte, et aucun doute n'est possible : un lien biologique est sousjacent. Par contre pour les scores un peu moins importants, il n'y a plus de corrélation entre le score et le Z-score, le Z-score apporte une information supplémentaire.

### 5.2.4 Z-scores et séquences aléatoires

Vingron et Waterman [128, 133] ont donné une approximation de la loi des scores par une méthode poissonnienne. La loi des scores dépend des longueurs des deux séquences et de deux paramètres qui, eux, dépendent du modèle choisi pour les séquences aléatoires :

$$P(SW \leq t) = e^{-\gamma mnp^t} = e^{-e^{log(\gamma mn) + t\,log(p)}}$$

Le Z-score est une normalisation du score sur un échantillon de scores construits sur des séquences ayant mêmes longueurs et mêmes distributions en acides aminés que les séquences initiales. Dès lors, la loi du Z-score devrait ne pas dépendre de la longueur et des compositions en acides aminés. Pour confirmer cette non-dépendance, nous construisons plusieurs échantillons de Z-scores,

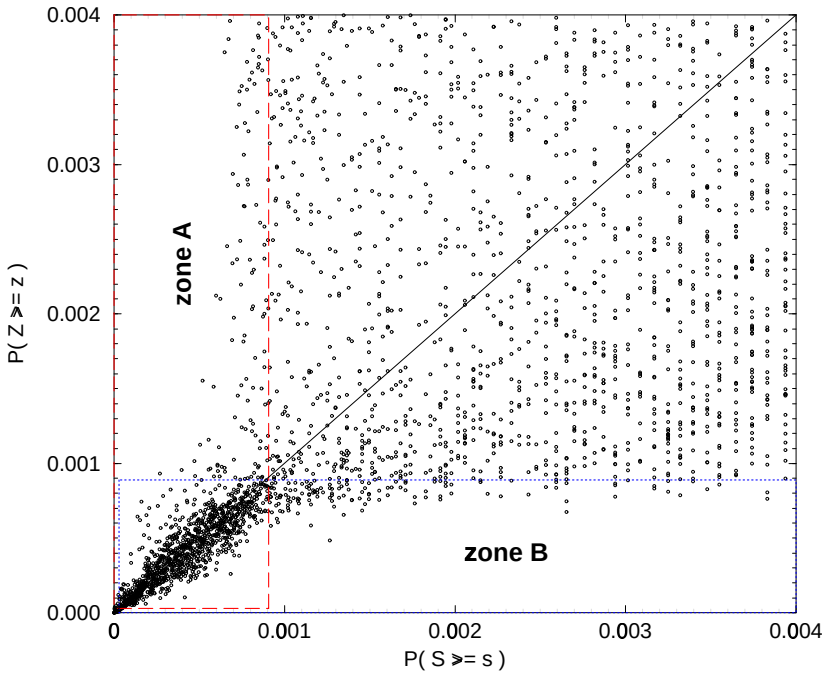



FIG. 5.4 - **Comparaison entre le Score et le Z-score :** *Pour toutes les paires de séquences du génome de* Haemophilus Influenzae, *on a calculé le score et le Z-score. Les probabilités empiriques du score* $P(S \geq s)$ *et du Z-score* $P(Z \geq z)$*ont été calculées. Pour chaque alignement on définit ces deux probabilités et on le représente dans le plan défini par ces probabilités. Si le Z-score et le score étaient équivalents, tous les points devraient être placés près de la diagonale. On observe une forte corrélation pour les petites probabilités (les scores et Z-scores sont élevés) mais une dispersion apparaît dans le voisinage du point* $(0.0009, 0.0009)$*, ce qui correspond à un Z-score de* $8.0$ *et un score de* $25$.

chacun obtenu sur des séquences aléatoires ayant des longueurs et compositions en acides aminés identiques. Pour plusieurs couples de séquences réelles nous effectuons les opérations suivantes :

1. Construction de deux banques de séquences aléatoires : l'une obtenue par permutation de la première séquence initiale, et l'autre obtenu en permutant la seconde. Chacune des deux banques comporte 500 séquences.

2. Calcul d'un échantillon de taille 500 de scores et de Z-scores ( première séquence de la première banque contre première séquence de la deuxième, puis seconde contre seconde, etc ). Les scores et Z-scores sont calculés avec la matrice Dayhoff (dont les coefficients ont été multipliés par 10/3), et avec les pénalités d'insertions/délétions 5 et 0.3. Le Z-score a été calculé avec 100 permutations pour chacune des deux séquences.

3. Calcul de la fonction de répartition empirique, à la fois pour le score et pour le Z-score.

Ces opérations sont effectuées pour des séquences de longueurs différentes :

| Séquence 1 | Séquence 2 | Long. 1 | Long. 2 |
|---|---|---|---|
| SLR1256 | HI0541 | 100 | 100 |
| SLR1958 | ECAE000486.YJDC | 200 | 200 |
| SLR1363 | HI0022 | 500 | 500 |
| YKR009C | YBR225W | 900 | 900 |
| YOL021C | SLR0744 | 1001 | 1001 |
| YOL021C | HI0541 | 1001 | 100 |

Il est à noter que chacune de ces séquences a une distribution en acides aminés spécifiques (Cf. figure 5.5). De plus, nous avons créé deux banques de 500 séquences aléatoires dont chaque séquence

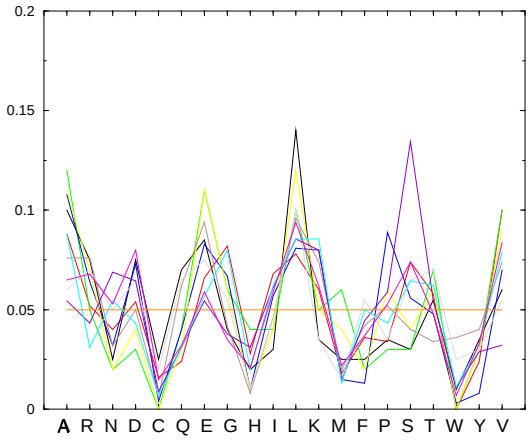



FIG. 5.5 - **Composition en acides aminés des différentes séquences.** *La ligne horizontale représente la distribution uniforme en acides aminés.*

est de longueur 100. Pour ces deux banques, la distribution en acides aminés est la distribution uniforme. Sur ces deux banques, de la même manière que précédemment, on construit un échantillon de scores et un échantillon de Z-scores.

La figure 5.6 donne les 7 fonctions de répartition obtenues pour les scores et les 7 fonctions associées aux Z-scores. Les scores aléatoires dépendent nettement des longueurs des deux séquences.

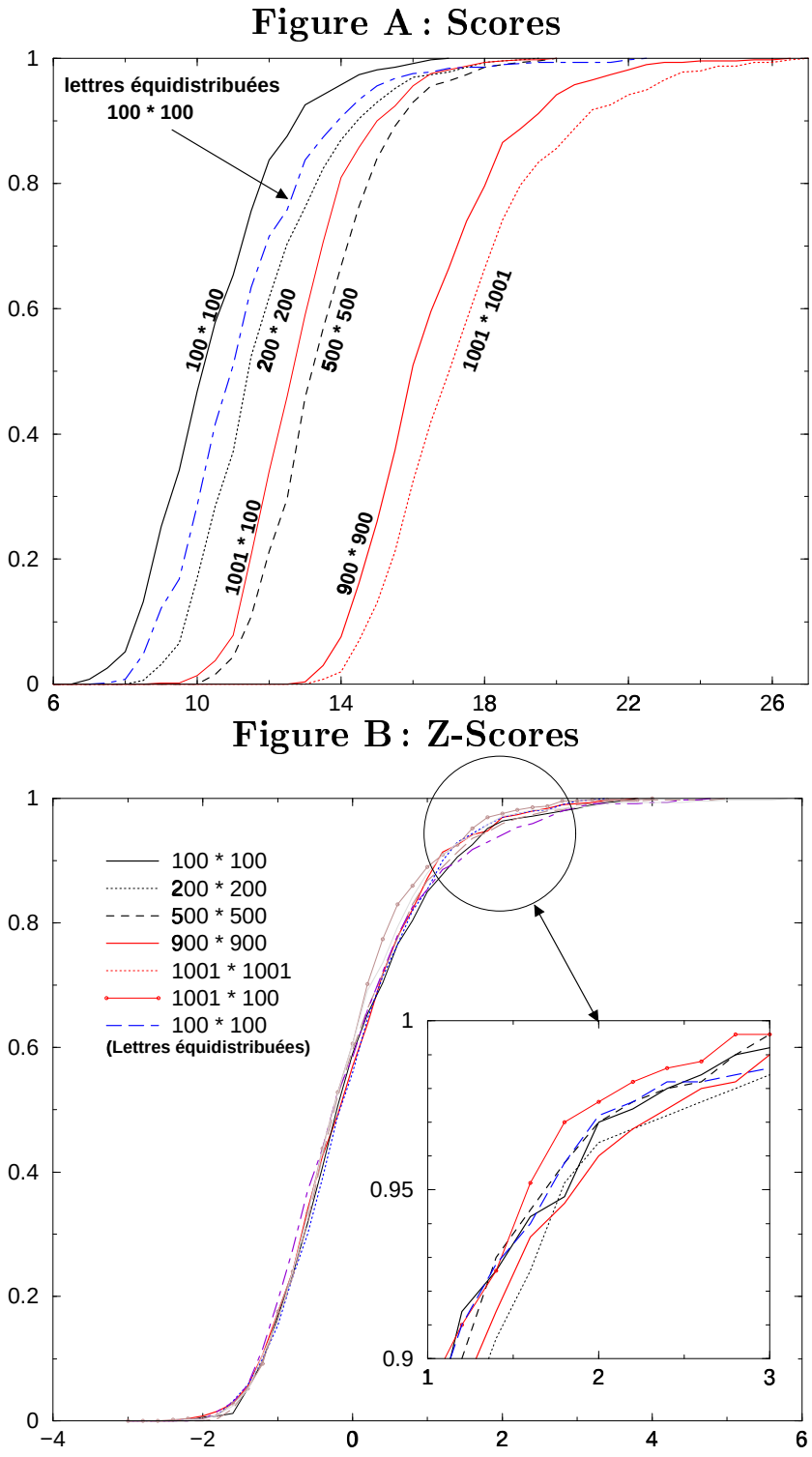



FIG. 5.6 - **Fonction de répartition du score et du Z-score sur des séquences aléatoires :** *La loi des scores dépend du produit des deux longueurs. Le Z-score réduit cette dépendance. Les produits définissant les courbes correspondent aux produits des longueurs des deux séquences intervenant dans le calcul.*

Dans le cas du Z-score, les sept courbes sont très proches malgré quelques variations pour les grandes valeurs.

La figure 5.6-A exprime la dépendance de la loi du score en fonction des longueurs des séquences impliquées. Plus le produit des longueurs des séquences est élevé, plus le score est élevé. La fonction de répartition la plus à droite correspond au produit des longueurs le plus élevé. Le modèle de Vingron & Waterman est qualitativement confirmé. De plus il y a bien une dépendance du score en fonction de la distribution en acides aminés. Pour des séquences de longueurs fixées à 100, lorsque les lettres sont équidistribuées, la courbe de la fonction de répartition du score se trouve déplacée vers la droite. Il est à noter que les courbes sont bien lisses malgré le faible nombre de scores (500) pour chaque répartition empirique.

La dépendance en longueur a été supprimée lors du passage au Z-score (Cf. figure 5.6-B). La dépendance en composition en acides aminés a aussi disparu. Il semble alors envisageable de donner une approximation de la loi du Z-score, qui ne dépende ni des longueurs des séquences, ni des compositions en acides aminés.

### 5.2.5 Influence de la longueur sur des Z-scores réels

Le paragraphe précédent montre que le score est dépendant de la longueur. Deux longues séquences qui ne sont pas biologiquement reliées, peuvent générer un score plus grand que deux petites séquences dont la relation biologique a été démontrée. On ne peut donc pas utiliser le score SW pour comparer deux alignements entre eux, ni l'utiliser comme filtre pour éliminer les alignements non significatifs.

Le Z-score, lui, reste indépendant de la longueur dans le cas des séquences aléatoires (Cf. figure 5.6-B). Dans le cas des séquences réelles, on n'a pas assez de séquences de longueurs fixes, pour pouvoir faire la même étude. On a donc été amené à calculer tous les scores et les Z-scores pour le génome complet de l'organisme *Methanococcus jannaschii*. Les alignements ont ensuite été regroupés suivant le produit des longueurs des deux séquences impliquées. Pour chaque intervalle du produit de longueurs, la moyenne et l'écart-type des scores et Z-scores obtenus ont été estimés. La figure 5.7 représente ces moyennes et écart-types pour l'organisme *Methanococcus jannaschii*.

L'observation sur ces données particulières illustre le comportement croissant du score en fonction de la longueur (confirmation de la dépendance du score en fonction de la longueur). Le Z-score est beaucoup plus stable et, même si une très légère augmentation du Z-score moyen peut être observée en fonction de la longueur, elle reste nettement inférieure à celle du score.

La difficulté de l'interprétation vient du fait qu'à l'intérieur du même génome, certaines séquences ont été dupliquées. Ainsi, certains scores et Z-scores sont statistiquement énormes, mais correspondent au phénomène commun de la duplication.

### 5.2.6 Loi des Z-scores

Nous avons essayé de déterminer la loi de probabilité des Z-scores. Une étude préliminaire indique que les scores et Z-scores semblent suivre une loi des valeurs extrêmes, la loi de Gumbel. Cela est à mettre en relation avec les travaux de Karlin, Altschul [5, 78, 79], qui ont montré que les scores BLAST suivent une loi de Gumbel (Cf. partie 1.3.1, théorème 1.3.1). On peut aussi citer les travaux de Mott [98] qui cherche une expression des coefficients de la loi de Gumbel dépendant de la longueur des séquences pour modéliser la loi des scores SW.

Puisque la dépendance en longueur du Z-score est nettement inférieure à celle du score, une loi du Z-score, indépendante de la longueur, devrait pouvoir être estimée.

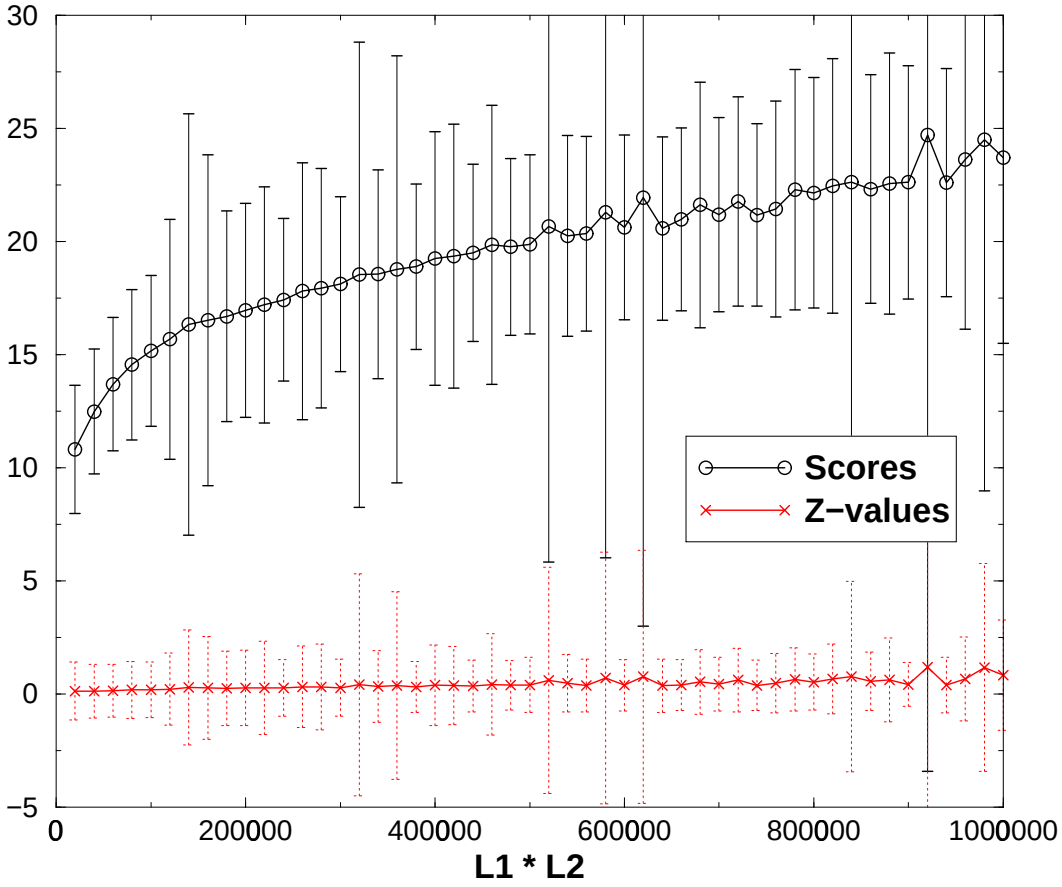



FIG. 5.7 - **Le Z-score diminue le biais en longueur :** *Pour le génome Methanococcus jannaschii, l'ensemble des scores et Z-scores ont été calculés. Les alignements sont regroupés en fonction du produit des longueurs des deux séquences impliquées. Pour chacun des intervalles du produit des deux longueurs (par pas de 20 000), les moyennes des scores et des Z-scores sont représentées. Les barres verticales représentent les écarts-types associés. Si le score est nettement dépendant de la longueur, le Z-score reste constant en moyenne.*

La loi de Gumbel est caractérisée par deux paramètres, la valeur caractéristique $\theta$ et la valeur de centrage $\xi$ :

$$P(Z \geq z) = 1 - exp(-exp^{-(z-\xi)/\theta})$$

où $z$ est la valeur observée du Z-score. La densité de la loi de Gumbel est :

$$p_Z(z) = \frac{1}{\theta} exp^{-(z-\xi)/\theta} exp(-exp^{-(z-\xi)/\theta}) \tag{5.6}$$

Il existe deux estimateurs possibles :

- Les estimateurs fondés sur les moments simples : $\tilde{\theta}$ et $\tilde{\xi}$. Soit $\mu$ et $\sigma$ la moyenne et l'écart-type empiriques de l'échantillon considéré. Ces estimateurs sont donnés par les formules [76] :

$$\tilde{\theta} = (\frac{\sqrt{6}}{\pi}).\sigma$$
$$\tilde{\xi} = \mu - \gamma.\tilde{\theta}$$

  où $\gamma$ est la constante d'Euler : $\gamma = 0.5772$.

- Les estimateurs du maximum de vraisemblance : $\hat{\theta}$ et $\hat{\xi}$. $\hat{\theta}$ est solution de l'équation suivante [76] :

$$\hat{\theta} - \mu + \left[\sum_{j=1}^{n} X_j exp^{-X_j/\hat{\theta}}\right] . \left[\sum_{j=1}^{n} exp^{-X_j/\hat{\theta}}\right]^{-1} = 0. \tag{5.7}$$

$$\hat{\xi} = \mu - \gamma.\hat{\theta} \tag{5.8}$$

  $\hat{\theta}$ et $\hat{\xi}$ sont des estimateurs biaisés de $\theta$ et $\xi$ [76], mais meilleurs que les estimateurs précédents. L'estimateur du maximum de vraisemblance $\hat{\theta}$ est initialisé avec la valeur de l'estimateur des simples moments $\tilde{\theta}$, et on utilise une procédure de Newton-Raphson pour déterminer la racine de l'équation 5.7. $\hat{\xi}$ peut ensuite être calculé.

Nous allons tester si le score et/ou le Z-score suit une loi des valeurs extrêmes. Pour cela, deux échantillons différents de 1 000 séquences ont été considérés.

- Le premier échantillon est un ensemble de 1 000 séquences réelles extraites de manière aléatoire du génome complet de la levure. A partir de ces séquences, tous les scores possibles ont été calculés (499 500 scores obtenus par l'algorithme de Smith & Waterman sur les 1000 séquences). Cet ensemble est appelé $\mathcal{S}(\mathcal{R})$. Puis, nous avons calculé l'ensemble $\mathcal{Z}(\mathcal{R})$ des 499 500 Z-scores.

- Le deuxième échantillon est construit par permutation de ces séquences réelles. Plus précisement, la séquence numéro $i$ de cette banque est une permutation de la séquence numéro $i$ de la banque de séquences réelles. Ces séquences sont considérées comme représentatives du hasard, en prenant en compte la distribution en acides aminés et la distribution en longueur des séquences. Sur cet échantillon de *séquences aléatoires*, les 499 500 scores et les 499 500 Z-scores ont été générés. Ces ensembles de scores et Z-scores sont appelés $\mathcal{S}(\mathcal{S})$ et $\mathcal{Z}(\mathcal{S})$.

Le score semble suivre à première vue une loi des valeurs extrêmes. Cependant, les tests du $\chi^2$ et de Kolmogorov-Smirnov ont refusé l'hypothèse nulle d'adéquation à la loi, et ceci pour différentes tailles de sous-échantillons. On se référera aux figures 5.8 et 5.9. Si on considère que le modèle

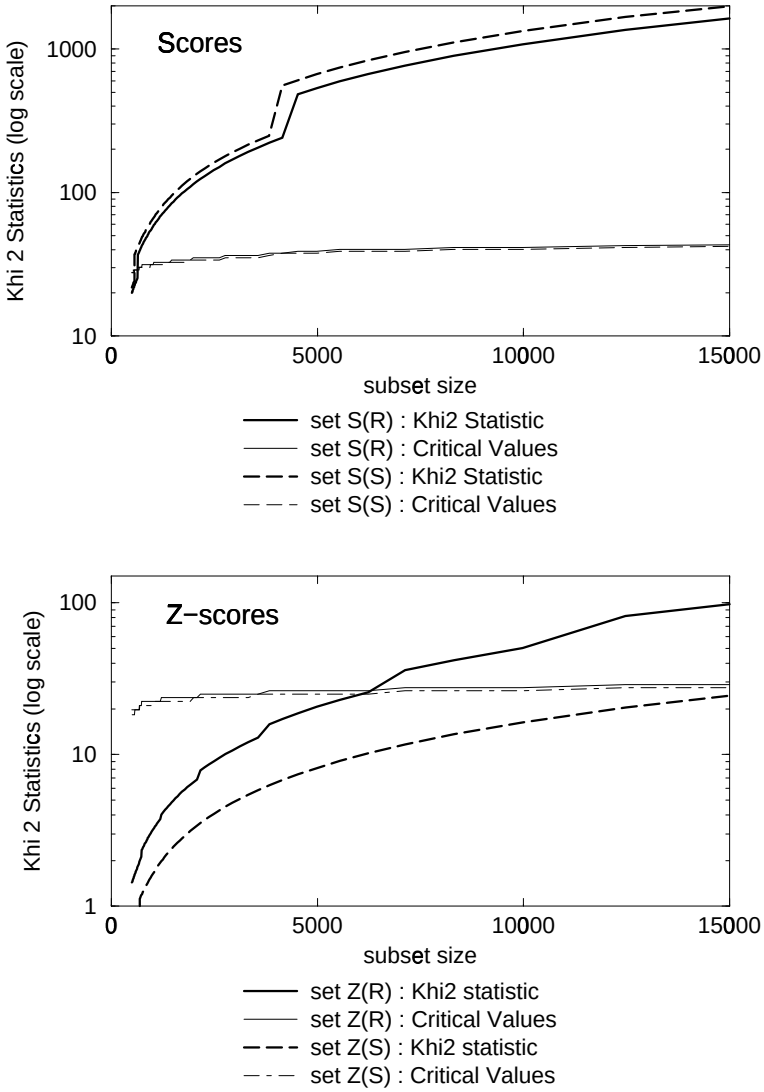



FIG. 5.8 - **Tests du $\chi^2$ :** *Pour des sous-échantillons de scores $\mathcal{S}(\mathcal{R})$ et $\mathcal{S}(\mathcal{S})$, et de Z-scores $\mathcal{Z}(\mathcal{R})$ et $\mathcal{Z}(\mathcal{S})$,de différentes tailles, la statistique du $\chi^2$ a été calculée. Le test rejette l'hypothèse d'adéquation à la loi des valeurs extrêmes, si la statistique du $\chi^2$ est plus grande que $\chi^2_\alpha(d)$ où $d$ est le nombre de degré de liberté. Les valeurs critiques $\chi^2_\alpha(d)$ sont représentées pour un niveau de confiance $\alpha = 5\%$.*

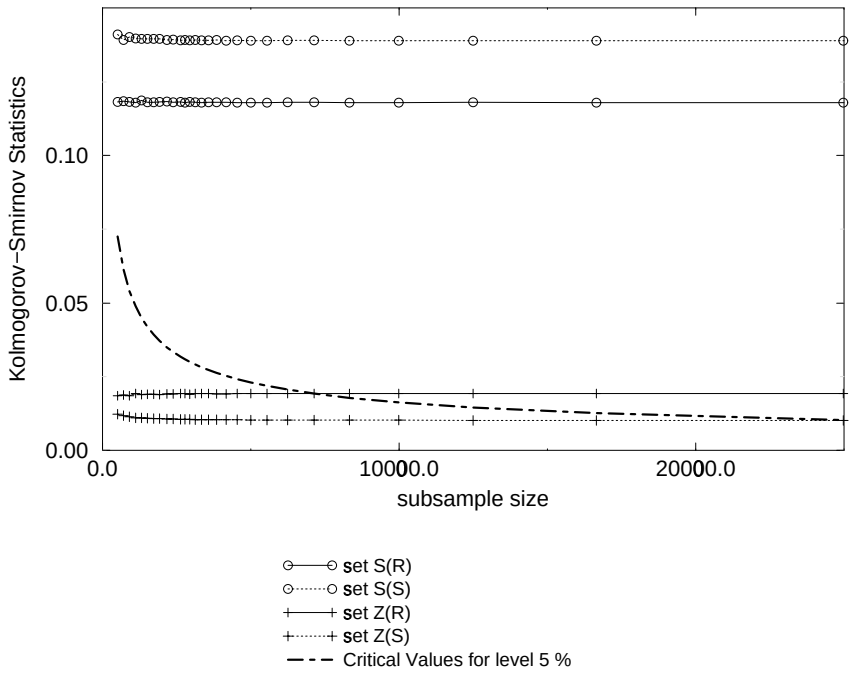



FIG. 5.9 - **Tests de Kolmogorov-Smirnov :** *Pour des sous-échantillons de scores et Z-scores, la statistique de Kolmogorov-Smirnov a été calculée. Le test rejette l'hypothèse d'adéquation à la loi des valeurs extrêmes, si la statistique observée est plus grande que la valeur critique. Les valeurs critiques sont représentées pour un niveau de confiance de 5%. Seuls les Z-scores semblent suivre la loi de Gumbel.*

de Waterman & Vingron est valide, les ensembles considérés de scores (sur lesquels on estime les paramètres de la loi de Gumbel) sont des mélanges de lois de Gumbel dont les paramètres dépendent des longueurs des séquences. Il n'est donc pas étonnant que ce mélange ne suive pas une loi de Gumbel. Il n'existe pas une loi de Gumbel modélisant l'ensemble des données.

Au contraire, les résultats sont plus satisfaisants pour le Z-score.

- Pour les séquences permutées, les résultats sont très bons puisque les tests acceptent l'hypothèse d'adéquation pour des sous-échantillons dont la taille va jusqu'à 16 000 dans le cas du test du $\chi^2$ et jusqu'à 20 000 dans le cas du test de Kolmogorov-Smirnov (la statistique est inférieure aux valeurs critiques). Le Z-score semble suivre la loi de Gumbel de la plus petite à la plus grande valeur.
- Pour les séquences réelles, les deux tests acceptent l'hypothèse d'adéquation à la loi, pour des sous-échantillons de tailles inférieures à 6 000. La loi des valeurs extrêmes est une bonne estimation de la loi du Z-score. Cependant, pour les grandes valeurs du Z-score, on observe une sur-représentation importante par rapport à la loi de Gumbel (Cf. figure 5.10).

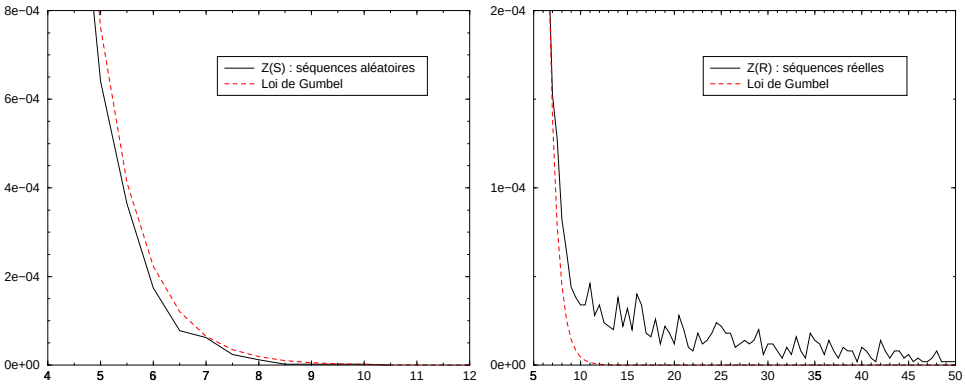



FIG. 5.10 - **Queue de distribution du Z-score :** *Alors que pour les séquences aléatoires la loi de Gumbel modélise le comportement du Z-score, dans le cas des séquences réelles, on observe une sur-représentation des grandes valeurs de Z-scores par rapport à la loi des valeurs extrêmes.*

Cette sur-représentation des grandes valeurs peut mener à une surévaluation de la significativité d'un alignement. Remarquons que cette sur-représentation n'est pas spécifique à l'ensemble de séquences considérées. Pour d'autres génomes, le même type de comportement est observé. La figure 5.11 montre que les courbes observées pour différents génomes divergent de la loi de Gumbel, à partir d'un certain seuil.

Deux ensembles de Z-scores ont été considérés : l'un est construit sur des séquences aléatoires et l'autre sur des séquences réelles. Dans les deux cas, la loi de Gumbel semble satisfaisante même si pour les séquences réelles, on observe pour les grands Z-scores un écart à la loi. Or, ce qui intéresse le biologiste, c'est justement les similarités qui ne sont pas dues au hasard. L'écart observé pour les séquences réelles correspond à ces similarités recherchées.

On s'intéresse donc à la recherche de la valeur à partir de laquelle les Z-scores réels divergent de la loi de Gumbel. Les Z-scores supérieurs à ce seuil seront considérés comme n'ayant pas été obtenus par chance, et, on fera l'hypothèse qu'un lien biologique entre les séquences impliquées est sous-jacent.

| Génome | Nombre de séquences | Nombre de comparaisons |
|---|---|---|
| All vs. All | 16 956 | 143 744 490 |
| Haemophilus influenzae | 1 680 | 1 410 360 |
| Methanococcus jannaschii | 1 735 | 1 504 245 |
| Synechocystis | 3 168 | 5 016 528 |
| S. cerevisiæ | 6 087 | 18 522 741 |
| Escherichia coli | 4 286 | 9 182 755 |
| YR | 1 000 | 499 500 |

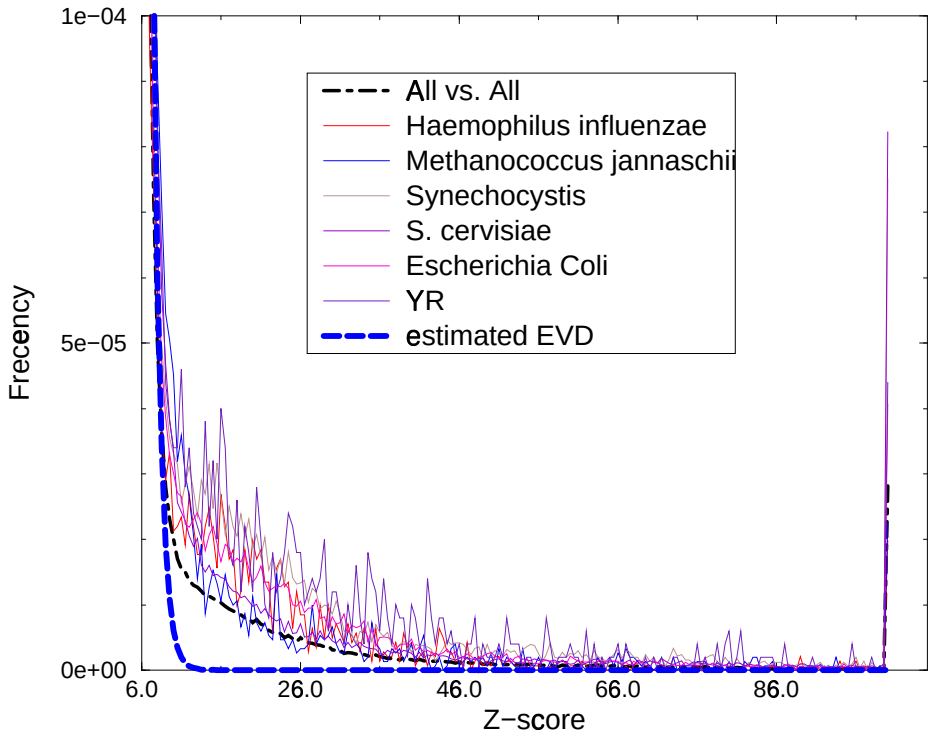



FIG. 5.11 - **Densité des Z-Scores.** *Pour différents génomes, la densité du Z-score a une queue de distribution non négligeable, qui diffère de la double exponentielle de la loi de Gumbel. En considérant que le Z-score suit une loi du type double exponentielle, nous surestimons la signification d'un Z-score.*

### 5.2.7 Détermination d'une valeur de *cutoff*

Pour déterminer la valeur de ce seuil, on étudie un ensemble de processus de Bernoulli. Pour chaque valeur de Z-score $c$, soit $N_{obs}^{c}$ le nombre observé de valeurs plus grandes que $c$. Considérons le processus de Bernoulli $X \longrightarrow \mathcal{B}\,(N, p_c)$, où $N$ est le nombre de Z-scores observés et $p_c$ est la probabilité pour que la variable $Z$ de type Gumbel soit plus grande que $c$ : $p_c = P_{Gumbel}(Z > c)$. Sous l'hypothèse de la loi de Gumbel, $X$ représente le nombre espéré de Z-scores supérieurs à $c$. Si le Z-score suit effectivement la loi de Gumbel, alors $N_{obs}^{c}$ a même loi que $X$. Pour une valeur de $c$, $N_{obs}^{c}$ peut être inférieur ou supérieur à $X$. Ces variations doivent être centrées sur la valeur obtenue sous le modèle théorique. On en déduit que si le Z-score suit effectivement la loi de Gumbel, alors la probabilité $P_{\mathcal{B}(N,p_c)}(X > N_{obs}^{c})$ doit être à peu près $\frac{1}{2}$, et ceci pour chaque valeur de $c$.

Pour des séquences aléatoires, la probabilité reste dans un intervalle correct. Au contraire, pour les séquences réelles, cette probabilité décroît très rapidement vers zéro. La valeur pour laquelle cette probabilité tombe à zéro est le seuil recherché. Comme le montre la figure 5.12, la distribution empirique des Z-scores s'écarte de la distribution des valeurs extrêmes à partir d'une valeur comprise entre 6.0 et 7.0. Nous considèrerons la valeur 8.0 pour une plus grande sécurité.

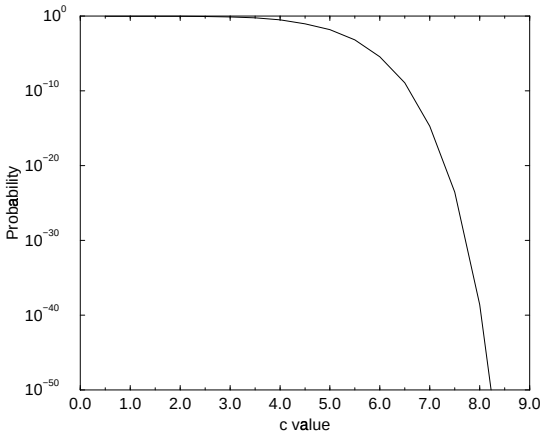



FIG. 5.12 - **Estimation de la valeur du seuil** *séparant les Z-scores qui suivent approximativement la distribution des valeurs extrêmes, des grands Z-scores. Soit $X \longrightarrow \mathcal{B}\,(N, p_c)$ une variable binômiale, où $N$ est le nombre de Z-scores observés, et $p_c$ la probabilité pour que la variable aléatoire $Z$ de type Gumbel, soit plus grande que $c$. $X$ est le nombre espéré de Z-scores supérieurs à $c$. La figure montre les variations de $P(X > N_{obs}^{c})$, où $N_{obs}^{c}$ est le nombre observé de Z-scores supérieurs à $c$.*

La même étude a été faite sur d'autres génomes complets : *Methanococcus jannaschii*. La valeur du seuil est là encore dans le même intervalle $[6.0; 8.0]$.

Regardons de nouveau la figure 5.4, p. 153. Si l'on se focalise uniquement sur les paires de séquences qui ont un Z-score supérieur à notre seuil 8.0 $\Big(P(Z \geq z) \leq 0.0009\Big)$, on sélectionne des paires qui n'ont pas forcément les scores les plus grands. Dans ce cas, on ne retient que les alignements de la zone $B$. Au contraire, les paires de séquences ayant un score supérieur à 25

$\Big(P(S \geq s) \leq 0.0009\Big)$, se situent dans la zone $A$ de la figure.

Si on ne retient que les alignements dont le Z-score dépasse 8, on sélectionne des alignements avec des scores pas très hauts, et on élimine certaines paires de séquences avec un grand score vraisemblablement dû à des distributions en acides aminés similaires ou dû à de grandes longueurs.

Nous avons essayé d'estimer la loi des Z-scores plus grands que 8.0. La figure 5.13 montre les distributions observées pour le Z-score sur des séquences réelles et des séquences aléatoires partageant les mêmes longueurs et les mêmes distributions en acides aminés. On y voit bien un changement de régime. Les Z-scores réels n'ont pas la même loi que les Z-scores obtenus sur des séquences aléatoires. On cherche à modéliser le phénomène biologique sous-jacent, qui génère ces Z-scores différents de ceux obtenus avec des séquences aléatoires.

On peut considérer qu'après le changement de régime observé, une droite en échelle Log-Log modélise bien le comportement des grands Z-scores (Cf. figure 5.13). Ceci nous amène à considérer le modèle dit de Pareto[3] [140]. Cette loi est représentée par une droite en échelle Log-Log.

Notons que dans la figure 5.13, on aperçoit une inflexion, dont le modèle de Pareto ne rend pas compte. Le changement de régime observé peut être interpreté par un changement des valeurs des paramètres de la loi de Gumbel plutôt que par un changement de la loi. Cependant, pour les données observées comprises entre 8 et 50, la loi de Pareto semble modéliser les données empiriques.

La densité de la loi de Pareto est :

$$\begin{array}{lcll} f(z) & = & \left\{ \begin{array}{ll} A.z^{-(1+\alpha)} & \text{si } z \geq 8.0 \\ 0 & \text{sinon} \end{array} \right. \\ avec & & \alpha \geq 0 \end{array}$$

Les paramètres $A$ et $\alpha$ dans la fonction $f(z)$ peuvent être calculés par une régression sur les Z-scores supérieurs à 8.0. Le coefficient $A$ est juste un coefficient de normalisation et n'apporte pas d'information supplémentaire. $\alpha$ est appelé l'indice de Pareto. Le coefficient $\alpha$ dans l'équation précédente a été calculé pour plusieurs génomes microbiens. Le tableau 5.1 résume les résultats obtenus.

### 5.2.8 Utilisation du QQ-plot

Cette technique (Probability integral transform) est surtout une méthode exploratoire. Pour toute variable aléatoire $X$ de fonction de répartition $F$ continue strictement croissante, la variable aléatoire $F(X)$ est uniforme sur l'intervalle $[0;1]$. En effet, $F(.)$ est une fonction strictement croissante donc inversible. $F^{-1}(y)$ est une fonction croissante sur $[0;1]$. En posant $y = F(x)$, on a

3. La loi de Pareto est quelquefois renommée à tort loi de *Zipf*, d'après le professeur George Kingsley Zipf (1902-1950) qui fit l'observation suivante. Le nombre des occurrences de certains événements classés par ordre de fréquence d'apparition, est déterminé par une fonction puissance du type $P(i) \approx \frac{1}{i^a}$.

L'exemple le plus connu est la fréquence des mots anglais. Le mot «the» est le plus courant classé numéro 1, suivi par les mots «of» et «to» classés numéro 2 et 3. Quand le nombre d'occurrences est ramené au rang des mots, on observe une loi de *Zipf*. Zipf donne aussi deux autres exemples : la population des villes et le revenu d'une entreprise. Ce dernier cas avait déjà été observé le siècle dernier par *Pareto*.

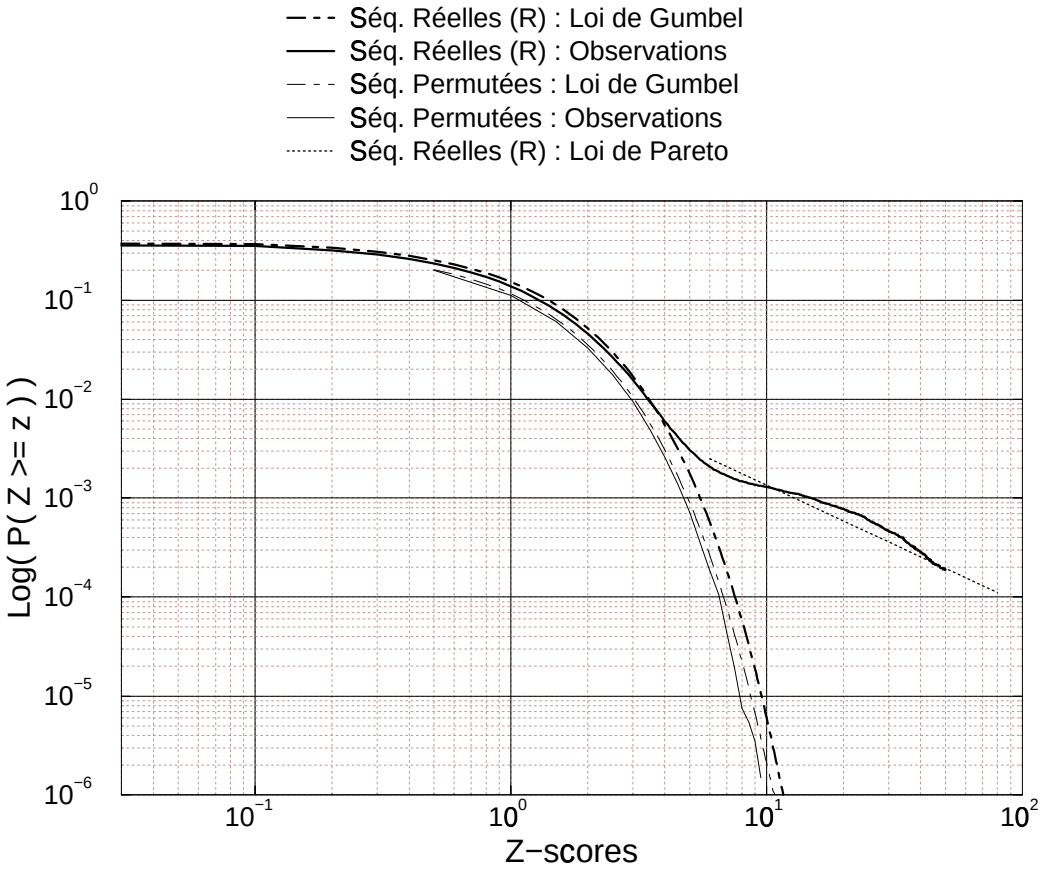



FIG. 5.13 - **Distributions des Z-scores en échelle Log-Log :** *La distribution des Z-scores sur les séquences permutées suit le modèle de Gumbel. Dans le cas des séquences réelles, on observe un biais à partir d'un Z-score de* 8. *A partir de ce seuil, la droite tracée correspond à une distribution de Pareto.*

| | Nombre de comparaisons | Coefficient de Pareto $\alpha$ |
|---|---|---|
| 1000 séquences de Yeast | 499500 | 1.20 |
| S. cerevisiæ | 18 522 741 | 0.90 |
| Escherichia coli | 9 182 755 | 1.26 |
| Haemophilus influenzae | 1 410 360 | 1.63 |
| Haemophilus influenzae (BLOSUM62) | 1 410 360 | 1.26 |
| Methanococcus jannaschii | 1 504 245 | 1.16 |
| Synechocystis | 5 016 528 | 1.05 |
| all vs. all | 143 744 490 | 1.16 |

TAB. 5.1 - **Coefficients de Pareto $\alpha$ pour différents génomes.** *moyenne* = 1.21, *écart-type* = 0.22. *Les coefficients de Pareto ont été calculés à partir des Z-scores (matrice = PAM250, gapo = 5, gape = 0.3) sauf pour* Haemophilus influenzae *où on a aussi calculé le coefficient avec la matrice BLOSUM62 et les paramètres gapo = 10, gape = 1.*

alors :

$$\begin{aligned} P(F(X) \leq y) &= P(X \leq F^{-1}(y)) \\ &= P(X \leq F^{-1}(F(x))) \\ &= P(X \leq x) \\ &= F(x) \\ &= y \end{aligned}$$

La fonction de répartition de la variable aléatoire $F(X)$ est la fonction identité sur l'intervalle $[0;1]$, et la variable aléatoire $F(X)$ est la loi uniforme sur $[0;1]$.

On se donne un échantillon $(Z_{i,N})_{i\in[1,N]}$ de taille $N$, classé par ordre croissant, de fonction de répartition $F$. On crée l'échantillon $(F_{i,N})_{i\in[1,N]} = \Big(F(Z_{i,N})\Big)_{i\in[1,N]}$. En classant l'échantillon, on a $F_{(1),N} \leq F_{(2),N} \leq \ldots \leq F_{(N),N}$. Puisque $F_{(i),N}$ est la $i^{ème}$ valeur d'un échantillon de taille $N$ de loi uniforme, on a $E\Big(F_{(i),N}\Big) = \frac{i}{N+1}$. Si la fonction de répartition de $Z$ est effectivement $F$, le nuage de point $\Big(\frac{i}{N+1}; F_{(i),N}\Big)$ doit s'accumuler autour de la première bissectrice. Ce graphique est appelé le *QQ-plot*.

Dans le cas de la loi de Gumbel, la fonction de répartition est bien strictement croissante puisque la densité n'est jamais nulle :

$$\begin{aligned} F(x) &= exp(-exp^{-(z-\xi)/\theta}) \\ F^{-1}(x) &= -\theta \times log\Big(-log(\frac{i}{N+1})\Big) + \xi \end{aligned}$$

Pour augmenter la précision, on utilise la transformation $F^{-1}$ et on regarde le nuage de points $\Big(Z_{i,N}; -\theta \times log\Big(-log(\frac{i}{N+1})\Big) + \xi\Big)$ (Cf. figure 5.14).

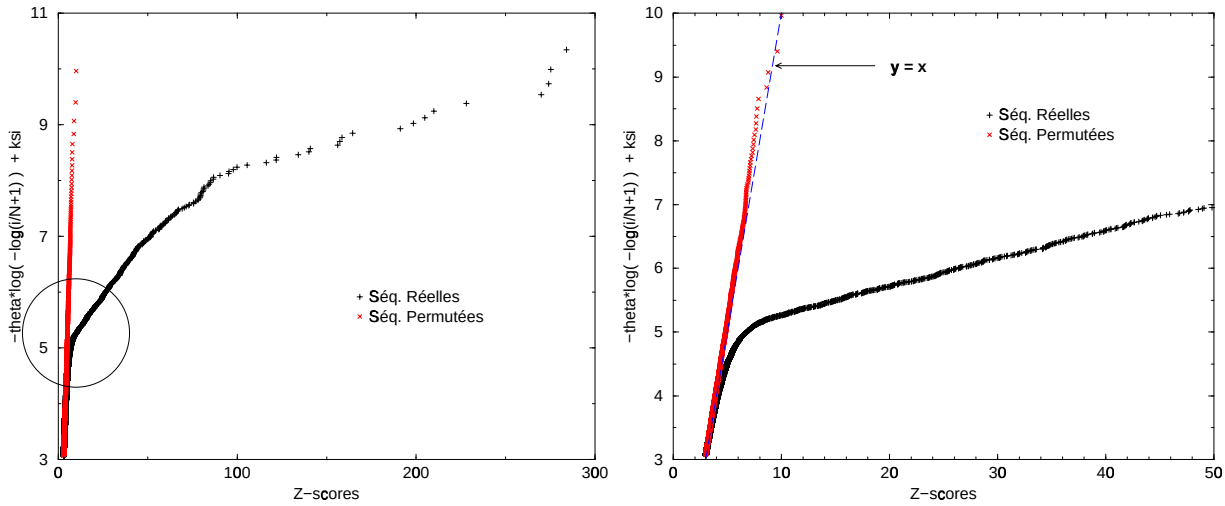



FIG. 5.14 - **QQ-plot sur deux ensembles de données issues de la levure.** *Les* 1000 *séquences réelles ont été choisies au hasard dans le génome complet de la levure, tandis que les séquences permutées sont des permutations de ces* 1000 *séquences. On obtient* 499500 *Z-scores pour les séquences réelles et autant pour les autres.*
*Le graphique de droite est une loupe de celui de gauche. Pour les séquences réelles,* 94 *données sont supérieures à* 50.

Dans la figure 5.14, on a tracé les QQ-plots pour deux ensembles de données issues du génome complet de la levure. Dans un premier temps, on s'intéresse aux séquences réelles, puis à des permutations des séquences réelles (Cf. paragraphe 5.2.6). Dans le cas des séquences permutées le nuage de points est très proche de la première bissectrice, ce qui confirme que la loi de Gumbel modèlise bien le comportement aléatoire du Z-score. Par contre dans le cas des séquences réelles, le nuage de points s'éloigne de la droite $y = x$ à partir d'une valeur comprise entre 4.0 et 6.0. Cette observation corrobore le choix du *cutoff* à 8.0.

Remarquons que si les paramètres sont mal estimés, le caractère linéaire du nuage de points n'est pas altéré. Les données se retrouvent alors autour d'une droite qui n'est plus la première bissectrice. Dans le cas où les paramètres sont inconnus, cette méthode peut donner une estimation des paramètres. Si la loi retenue modélise bien le comportement observé de l'échantillon, les points du diagramme s'accumulent sur une droite, dont les paramètres sont ceux de la loi, à une transformation près. Par exemple, si on sait que l'échantillon suit une loi de Gumbel de paramètres inconnus, les coefficients de la droite de régression du nuage de points dans le QQ-plot pour une loi de Gumbel de paramètres $(\theta = 1, \xi = 0)$ donne, à une transformation près, une première estimation des paramètres de la loi.

Ainsi, puisqu'on observe une droite pour les Z-scores compris entre 8 et 50, une loi de Gumbel modèlise bien le comportement du Z-score dans cet intervalle. Les coefficients de la loi de Gumbel pour cet intervalle peuvent être déterminés par regression linéaire sur le nuage de points. La table 5.2 donne les valeurs de ces coefficients pour différents génomes. Pour toutes les valeurs données, le coefficient de corrélation est supérieur à 0.977

**Discussion :**

Reprenons notre démarche. La loi du Z-score semble être une loi de Gumbel. Pour les Z-scores

| | *Nombre de comparaisons* | *EVD* | |
|---|---|---|---|
| | | $\xi$ | $\theta$ |
| 1000 séquences de Yeast | 499500 | −122.687 | 19.938 |
| S. cerevisiæ | 18 522 741 | −162.46 | 23.8 |
| Escherichia coli | 9 182 755 | −119.889 | 18.501 |
| Haemophilus influenzae | 1 410 360 | −93.436 | 14.448 |
| Haemophilus influenzae (BLOSUM62) | 1 410 360 | −124.226 | 19.363 |
| Methanococcus jannaschii | 1 504 245 | −127.938 | 18.682 |
| Synechocystis | 5 016 528 | −135.259 | 21.498 |
| all vs. all | 143 744 490 | −136.921 | 19.085 |

TAB. 5.2 - **Coefficients de la loi de Gumbel** *pour les Z-scores compris dans l'intervalle* $[8; 50]$. *Tous les Z-scores ont été calculés avec la matrice PAM250 (gapo = 5, gape = 0.3). Pour le génome* Haemophilus influenzae*, les Z-scores ont été recalculés en utilisant la matrice BLOSUM62 (gapo = 10, gape = 1).*

permutés, cette loi modèlise bien les observations. Par contre, pour les Z-scores réels, une surreprésentation des grandes valeurs est observées. Pour modéliser le phénomène biologique qui génère ces grands Z-scores, deux modèles ont été envisagés : le modèle de Pareto, et un autre modèle de Gumbel. Passer du modèle de Gumbel pour les petits Z-scores à un modèle de Pareto pour les grands Z-scores n'est pas complètement satisfaisant.
La rupture observée est plus vraisemblablement due à un changement de paramètres de la loi de Gumbel.

Cette rupture peut être due à une mauvaise estimation de la moyenne et de l'écart-type. En effet, lors des permutations pour le calcul du Z-score, on n'effectue qu'une centaine de permutations. Dans le cas des petits Z-scores, ou des Z-scores permutés, les séquences permutées successives sont représentatives de l'ensemble des séquences qu'on peut obtenir par permutation. L'estimation empirique de la moyenne et de l'écart-type est bonne.
Dans le cas des grands Z-scores, les deux séquences se ressemblent, et les 100 séquences permutées ne permettent plus d'obtenir de bonnes estimations de la moyenne et de la variance.

Ce biais d'estimation des deux paramètres ne peut engendrer un changement de modèle. Nous retiendrons donc pour la loi du Z-score le modèle de Gumbel.

## 5.3 Application du Z-score : le Z-score est-il vraiment un meilleur indice de similarité ?

Considérons le problème classique de la recherche de similarité d'une séquence et d'une famille de séquences déjà établie. Ce genre de problème à été largement traité par un grand nombre de méthodes. On peut citer les HMM[4] [14][23][36][83][100], l'alignement entre une séquence et un

4. Hidden Markov Model.

alignement multiple [103], ou encore l'alignement avec un *profile*[5] [55, 57]. On va ici utiliser un indice simple de similarité entre une séquence et une famille. Dans ce problème, il n'est pas nécessaire que les séquences soient alignées au préalable.

On se donne une famille $\mathcal{F} = \{F_1, F_2 \dots F_k\}$ contenant $k$ séquences, et une séquence extérieure $X$ dont on se demande si elle ressemble à la famille $\mathcal{F}$. On se donne en plus un indice de similarité entre 2 séquences $A$ et $B$ : $s(A, B)$. On définit deux indices de similarité de $X$ à $\mathcal{F}$ :

$$\mathcal{S}(X, \mathcal{F}) = \frac{1}{k} \sum_{i=1}^{k} s(X, F_i) \tag{5.9}$$

$$\mathcal{S}'(X, \mathcal{F}) = \max_{i=1}^{k} s(X, F_i) \tag{5.10}$$

On compare les résultats obtenus pour différents indices de similarité entre deux séquences.

**Descriptions des données :**

- Prenons une famille bien connue de protéines : les globines. On prend comme famille la moitié des séquences du domaine n$^o$ 1 de ProDom 28 [119]. On a 302 séquences.
- On prend pour séquences extérieures à la famille
  - les 66 globines de SwissProt Rel. 32 qui n'appartiennent pas au domaine de ProDom,
  - des protéines qui ne sont pas des globines tirées au hasard dans SwissProt Rel. 32 (1875 séquences).
- Les indices de similarité par paire retenus sont les suivants :
  - le score du meilleur alignement local sans gap trouvé par BLAST (PAM250),
  - le score Smith & Waterman (PAM250, $gapo = 10$, $gape = 1$),
  - le Z-score (PAM250, $gapo = 10$, $gape = 1$).

Les tableaux suivants (5.5 et 5.6) montrent les avantages d'une méthode prenant en compte des insertions/délétions, mais permettent aussi de donner un avantage au Z-score par rapport au score.

**Discussion :** Lorsqu'on regarde de plus près les globines, et plus précisément le domaine de ProDom concerné, on se rend compte qu'il y a des sous-familles qui se différencient par l'apparition d'insertions/délétions. Il n'est donc pas du tout étonnant que la méthode avec insertions/délétions donne de meilleurs résultats, en diminuant le nombre de faux négatifs.

De plus, prendre la moyenne des scores plutôt que le maximum des scores permet de rejeter plus facilement les similitudes dues, non pas aux caractéristiques de la famille, mais à la spécificité d'une ou plusieurs séquences de la famille.

Nous avons extrait de cette famille de globines un sous-ensemble de taille 10 (Cf. figure 5.15). Bien que ces 10 séquences soient très proches et qu'elles ne représentent pas l'ensemble de la famille, les résultats sont très bons (Cf. tableau 5.3). Parmi les 66 séquences les plus similaires à cette sous-famille (avec la moyenne des scores SW), il n'y en a que 22 qui ne sont pas des globines, et la première non-globine apparaît au rang 28. Le rang moyen des globines est 181. Le Z-score (100

5. Un profile est défini par les données des probabilités d'apparition de chacun des acides aminés pour chaque position. Un profile peut servir à la définition d'une famille.

```
HBB_LUTLU  19   145  ......NVDEVGGEALGR..LLVV...YPWTQRFF.DSFGDLSSP..VMGNPKVKAHGK.KVLNSFSEGLK....
HBB_PHOVI  19   145  ......NVDEVGGEALGR..LLVV...YPWTQRFF.DSFGDLSSA..IMGNPKVKAHGK.KVLNSFSDGLK....
HBB_PTEBR  19   145  ......NVDEVGGEALGR..LLVV...YPWTQRFF.DSFGDLSSP..VMGNPKVKAHGK.KVLNSFSEGLK....
HBB_LEPWE  19   133  ......NVDEVGGEALGR...LLV..VYPWTQRFF.DSFGDLSSP..IMSNPKVKAHGK.KVLNSFSDGLK....
HBB_ODORO  19   145  ......NVDEVGGEALGR..LLVV...YPWTQRFF.DSFGDLSSP..VMGNPKVKAHGK.KVLNSFSDGLK....
HBB_SHEEP  18   144  ......KVDEVGAEALGR..LLVV...YPWTQRFF.EHFGDLSNA..VMNNPKVKAHGK.KVLDSFSNGMK....
HBB_ERIEU  19   133  ......KVEEFGGEALGR..LLVV...YPWTQRFF.DSFGDLSSA..VMGNPKVKAHGA.KVLQSMGDGIK....
HBB_NASNA  19   145  ......NVDEVGGEALGR..LLVV...YPWTQRFF.ESFGDLSSP..IMGNPKVKAHGK.KVLNSFSEGLK....
HBB_CYNSP  19   132  ......KVDEVGGEALGR..LLVV...YPWTQRFF.DSFGDLSSA..VMGNAKVKAHGK.KVLDSFSEGLQ....
HBB_TUPGL  19   145  ......DLEKVGGQSLGS...LLI..VYPWTQRFF.DSFGDLSSP..VMSNPKVKAHGK.KVLTSFSDGLN....

HBB_LUTLU            NLDNLKGTFAKLSELHCDKLHVDPENFKLLGNVLVCVL.AHHFG.KEFTPQVQAAYQKV.VAGVANALAHKY
HBB_PHOVI            NLDNLKGTFAKLSELHCDKLHVDPENFKLLGNVLVCVL.AHHFG.KEFTPQVQAAYQKV.VAGVANALAHKY
HBB_PTEBR            NLDNLKGTFAKLSELHCDKLHVDPENFKLLGNVLVCVL.AHHFG.KEFTPQVQAAYQKV.VAGVANALAHKY
HBB_LEPWE            NLDNLKGTFAKLSELHCDQLHVDPENFKLLGNVLVCVLAHHFGK..EFTPQVQAAYQKV.............
HBB_ODORO            NLDNLKGTFAKLSELHCDKLHVDPENFKLLGNVLVCVL.AHHFG.KEFTPQVQAAYQKV.VAGVANALAHKY
HBB_SHEEP            HLDDLKGTFAQLSELHCDKLHVDPENFRLLGNVLVVVL.ARHHG.NEFTPVLQADFQKV.VAGVANALAHKY
HBB_ERIEU            NLDNLKGTFSKLSELHCDKLHVDPENFRLLGNVLVCVL.ARHFG.KDFTPAAQAAFQKV.............
HBB_NASNA            NLDNLKGTFAKLSELHCDKLHVDPENFRLLGNVLVCVL.AHHFG.KEFTPQVQAAY.QKVVAGVANALAHKY
HBB_CYNSP            HLDSLKGTFAKLSELHCDKLHVDPENFRLLGNVLVVVLARHFGK...EFTPQLQAAYQK.............
HBB_TUPGL            HLDNLKGTFAKLSELHCDKLHVDPENFRLLGNVLVRVL.ACNFG.PEFTPQVQAAFQKV.VAGVANALAHKY
```

FIG. 5.15 - **10 globines du domaine n$^o$ 1 de Prodom 28**

| | **Z-scores** | | **Scores** | |
|---|---|---|---|---|
| | moyenne | max | moyenne | max |
| rang moyen des globines | 151.6 | 170.5 | 181.1 | 201.2 |
| rang de la $1^{ère}$ non-globine | 41 | 41 | 28 | 28 |
| nombre de non-globines parmi les 66 meilleurs scores | 15 | 15 | 22 | 22 |

TAB. 5.3 - **Reconnaissance des globines à partir d'une petite famille de** 10 **globines.** *Le Z-score est calculé ici avec 100 permutations de chacune des 2 séquences*

permutations) améliore les résultats, puisque la première non-globine apparaît au rang 41, et qu'il n'y a que 15 non-globines parmi les 66 meilleurs résultats. Lorsqu'on prend le maximum au lieu de la moyenne, les résultats sont presque aussi bons : il n'y a pas plus de faux positifs, c'est-à-dire de non-globines parmi les 66 meilleurs résultats. La différence ne concerne que les 26 globines qui ont été classées après le premier faux positif (rang 41). Le rang de ces globines est plus élevé lorsqu'on prend le maximum.

Que devient ce classement lorsque le Z-score est calculé avec seulement 10 permutations ? Les résultats (Cf. tableau 5.4) sont certes moins bons, mais la dégradation n'est pas alarmante. Le rang de la première non-globine est passé de 41 à 30, mais le nombre de faux positifs n'a pas augmenté.

Le Z-score améliore sensiblement la recherche de similarité d'une séquence à une famille de séquences. L'amélioration porte sur le nombre de faux positifs, sur le rang du premier faux positif, mais aussi sur le rang moyen des séquences assimilées à la famille.

| | **Z-scores** | |
|---|---|---|
| | moyenne | max |
| rang moyen des globines | 153.6 | 183.2 |
| rang de la $1^{ère}$ non-globine | 30 | 24 |
| nombre de non-globines parmi les 66 meilleurs scores | 15 | 19 |

TAB. 5.4 - **Reconnaissance des globines à partir d'une petite famille de 10 globines.** *Le Z-score est calculé ici avec 10 permutations de chacune des 2 séquences.*

Mais l'intérêt majeur du Z-score réside dans le fait qu'il est indépendant des longueurs et des compositions en acides aminés, contrairement au score Smith & Waterman. Il est difficile de classer les alignements entre eux à partir du simple score. La classification des séquences au sein d'un génome qui nécessite la comparaison des alignements entre eux, ne peut donc pas se fonder sur le score puisque chaque score a une loi différente. Le Z-score est lui indépendant des longueurs des séquences, il est alors possible de l'utiliser à des fins d'analyse de génomes complets.

| nom de la séquence | | Z-scores | | Scores | | BLAST |
|---|---|---|---|---|---|---|
| | | moyenne | max | moyenne | max | moyenne |
| HBA1_ARCGA | * | 1 | 1 | 5 | 5 | 5 |
| HBA2_ARCGA | * | 2 | 2 | 4 | 6 | 4 |
| HBB_LYNLY | * | 3 | 3 | 1 | 2 | 1 |
| HBA_LYNLY | * | 4 | 6 | 6 | 4 | 6 |
| HBB_MICGA | * | 5 | 5 | 3 | 7 | 3 |
| HBB_APTFO | * | 6 | 4 | 2 | 3 | 2 |
| HBA_MICGA | * | 7 | 7 | 7 | 11 | 8 |
| GLB1_GLYDI | * | 8 | 9 | 8 | 17 | 264 |
| GLB4_GLYDI | * | 9 | 10 | 10 | 16 | 535 |
| GLB3_TYLHE | * | 10 | 23 | 15 | 29 | 313 |
| GLB3_LAMSP | * | 11 | 20 | 18 | 40 | 464 |
| GLB7_ARTSX | * | 12 | 41 | 9 | 39 | 912 |
| GLB4_LUMTE | * | 13 | 26 | 12 | 44 | 47 |
| GLB3_LUMTE | * | 14 | 32 | 11 | 31 | 223 |
| GLBT_CHITH | * | 15 | 13 | 13 | 15 | 15 |
| GLB1_PHESE | * | 16 | 24 | 14 | 22 | 347 |
| GLB3_LUCPE | * | 17 | 12 | 17 | 18 | 13 |
| LGB3_MEDSA | * | 18 | 43 | 24 | 28 | 82 |
| LGB4_MEDSA | * | 19 | 22 | 20 | 27 | 155 |
| GLB2_LUCPE | * | 20 | 37 | 16 | 21 | 14 |
| HBF1_URECA | * | 21 | 28 | 22 | 36 | 290 |
| HBPL_PARAD | * | 22 | 35 | 25 | 24 | 43 |
| GLB2_LUMTE | * | 23 | 34 | 27 | 37 | 463 |
| HBP2_CASGL | * | 24 | 21 | 19 | 25 | 60 |
| HBP1_CASGL | * | 25 | 30 | 26 | 35 | 670 |
| LGBA_PHAVU | * | 26 | 36 | 33 | 46 | 489 |
| LGB1_MEDSA | * | 27 | 17 | 28 | 33 | 148 |
| LGB2_MEDTR | * | 28 | 40 | 35 | 34 | 107 |
| HBPL_TRETO | * | 29 | 18 | 34 | 19 | 48 |
| GLB2_TYLHE | * | 30 | 53 | 21 | 71 | 229 |
| GLB_TUBTU | * | 31 | 46 | 29 | 43 | 936 |
| GLB1_LUCPE | * | 32 | 31 | 42 | 20 | 32 |
| LGB2_LUPLU | * | 33 | 16 | 37 | 32 | 288 |
| GLB1_TYLHE | * | 34 | 27 | 39 | 26 | 20 |
| GLB4_TYLHE | * | 35 | 56 | 36 | 51 | 67 |
| LGB1_MEDTR | * | 36 | 47 | 41 | 42 | 72 |
| LGB3_SOYBN | * | 37 | 25 | 44 | 58 | 615 |
| GLB1_LUMTE | * | 38 | 54 | 23 | 59 | 531 |
| LGB2_SOYBN | * | 39 | 51 | 43 | 66 | 898 |
| LGBA_SOYBN | * | 40 | 55 | 47 | 70 | 938 |
| LGB1_SOYBN | * | 41 | 49 | 50 | 80 | 995 |
| LGB_PSOTE | * | 42 | 29 | 40 | 54 | 664 |
| HMPA_ALCEU | * | 43 | 42 | 31 | 49 | 57 |
| HMPA_ECOLI | * | 44 | 45 | 30 | 30 | 203 |
| LGB1_PEA | * | 45 | 63 | 48 | 62 | 281 |
| FHP_CANNO | * | 46 | 58 | 32 | 76 | 55 |
| LGB3_SESRO | * | 47 | 44 | 45 | 75 | 282 |
| FHP_YEAST | * | 48 | 61 | 38 | 95 | 33 |
| LGB1_LUPLU | * | 49 | 48 | 51 | 86 | 996 |
| LGB2_SESRO | * | 50 | 39 | 49 | 92 | 265 |
| LGB1_VICFA | * | 51 | 57 | 68 | 67 | 316 |
| GLB1_CALSO | * | 52 | 62 | 65 | 38 | 702 |
| GLB2_CALSO | * | 53 | 59 | 92 | 133 | 890 |
| HMPA_VIBPA | * | 54 | 60 | 52 | 63 | 117 |
| APA1_RABIT | | 55 | 115 | 74 | 429 | 698 |
| MUP6_MOUSE | | 56 | 85 | 105 | 615 | 610 |
| MLRB_CHICK | | 57 | 200 | 189 | 616 | 523 |
| YFF7_YEAST | | 58 | 191 | 339 | 848 | 1340 |
| GLBH_CAEEL | * | 59 | 52 | 254 | 145 | 854 |
| PHYB_SOLTU | | 60 | 103 | 54 | 274 | 34 |
| NIF2_METIV | | 61 | 68 | 277 | 613 | 923 |
| GR78_PHAVU | | 62 | 227 | 1786 | 1792 | 1660 |
| GLB_ASCSU | * | 63 | 50 | 98 | 122 | 231 |
| 2AAB_HUMAN | | 64 | 65 | 71 | 407 | 770 |
| SYK_BACSU | | 65 | 184 | 64 | 60 | 465 |
| YCH2_YEAST | | 66 | 284 | 499 | 986 | 1224 |
| rang moyen des globines | | 88.5 | 109.9 | 166 | 180.7 | 409.3 |

TAB. 5.5 - **Comparaison entre une famille de globines et un ensemble de protéines (66 globines et 1875 non-globines).** *Le meilleur indice de similarité entre deux séquences est le Z-score. On a reporté les rangs (pour les différents indices de similarité) des 66 séquences apparentées à la famille en utilisant comme indice de similarité la moyenne des Z-scores. Les «*» marquent les globines.*

| rang | Z-scores | | | | Scores | | | | Scores BLAST | |
|---|---|---|---|---|---|---|---|---|---|---|
| | moyenne | | max | | moyenne | | max | | moyenne | |
| 1 | HBA1_ARCGA | * | HBA1_ARCGA | * | HBB_LYNLY | * | CBG_HUMAN | | HBB_LYNLY | * |
| 2 | HBA2_ARCGA | * | HBA2_ARCGA | * | HBB_APTFO | * | HBB_LYNLY | * | HBB_APTFO | * |
| 3 | HBB_LYNLY | * | HBB_LYNLY | * | HBB_MICGA | * | HBB_APTFO | * | HBB_MICGA | * |
| 4 | HBA_LYNLY | * | HBB_APTFO | * | HBA2_ARCGA | * | HBA_LYNLY | * | HBA2_ARCGA | * |
| 5 | HBB_MICGA | * | HBB_MICGA | * | HBA1_ARCGA | * | HBA1_ARCGA | * | HBA1_ARCGA | * |
| 6 | HBB_APTFO | * | HBA_LYNLY | * | HBA_LYNLY | * | HBA2_ARCGA | * | HBA_LYNLY | * |
| 7 | HBA_MICGA | * | HBA_MICGA | * | HBA_MICGA | * | HBB_MICGA | * | CBG_HUMAN | |
| 8 | GLB1_GLYDI | * | CBG_HUMAN | | GLB1_GLYDI | * | PEDF_HUMAN | | HBA_MICGA | * |
| 9 | GLB4_GLYDI | * | GLB1_GLYDI | * | GLB7_ARTSX | * | OVAL_COTJA | | PEDF_HUMAN | |
| 10 | GLB3_TYLHE | * | GLB4_GLYDI | * | GLB4_GLYDI | * | A1AT_BOMMO | | TGF3_CHICK | |
| 11 | GLB3_LAMSP | * | PEDF_HUMAN | | GLB3_LUMTE | * | HBA_MICGA | * | COLL_HSVS7 | |
| 12 | GLB7_ARTSX | * | GLB3_LUCPE | * | GLB4_LUMTE | * | SERA_MANSE | | A1AT_BOMMO | |
| 13 | GLB4_LUMTE | * | GLBT_CHITH | * | GLBT_CHITH | * | TGF3_CHICK | | GLB3_LUCPE | * |
| 14 | GLB3_LUMTE | * | TGF3_CHICK | | GLB1_PHESE | * | SPI1_COWPX | | GLB2_LUCPE | * |
| 15 | GLBT_CHITH | * | OVAL_COTJA | | GLB3_TYLHE | * | GLBT_CHITH | * | GLBT_CHITH | * |
| 16 | GLB1_PHESE | * | LGB2_LUPLU | * | GLB2_LUCPE | * | GLB4_GLYDI | * | E13D_HORVU | |
| 17 | GLB3_LUCPE | * | LGB1_MEDSA | * | GLB3_LUCPE | * | GLB1_GLYDI | * | DLD1_PSEPU | |
| 18 | LGB3_MEDSA | * | HBPL_TRETO | * | GLB3_LAMSP | * | GLB3_LUCPE | * | RBL_THEPO | |
| 19 | LGB4_MEDSA | * | SPI1_COWPX | | HBP2_CASGL | * | HBPL_TRETO | * | SERA_MANSE | |
| 20 | GLB2_LUCPE | * | GLB3_LAMSP | * | LGB4_MEDSA | * | GLB1_LUCPE | * | GLB1_TYLHE | * |
| 21 | HBF1_URECA | * | HBP2_CASGL | * | GLB2_TYLHE | * | GLB2_LUCPE | * | BCHX_RHOCA | |
| 22 | HBPL_PARAD | * | LGB4_MEDSA | * | HBF1_URECA | * | GLB1_PHESE | * | SPI1_COWPX | |
| 23 | GLB2_LUMTE | * | GLB3_TYLHE | * | GLB1_LUMTE | * | CLAB_LYCES | | CA1F_HUMAN | |
| 24 | HBP2_CASGL | * | GLB1_PHESE | * | LGB3_MEDSA | * | HBPL_PARAD | * | MIAA_ECOLI | |
| 25 | HBP1_CASGL | * | LGB3_SOYBN | * | HBPL_PARAD | * | HBP2_CASGL | * | MD10_YEAST | |
| 26 | LGBA_PHAVU | * | GLB4_LUMTE | * | HBP1_CASGL | * | GLB1_TYLHE | * | HD_MOUSE | |
| 27 | LGB1_MEDSA | * | GLB1_TYLHE | * | GLB2_LUMTE | * | LGB4_MEDSA | * | PTWC_ECOLI | |
| 28 | LGB2_MEDTR | * | HBF1_URECA | * | LGB1_MEDSA | * | LGB3_MEDSA | * | PFLB_HAEIN | |
| 29 | HBPL_TRETO | * | LGB_PSOTE | * | GLB_TUBTU | * | GLB3_TYLHE | * | L301_BOMMO | |
| 30 | GLB2_TYLHE | * | HBP1_CASGL | * | HMPA_ECOLI | * | HMPA_ECOLI | * | POLG_EMCV | |
| 31 | GLB_TUBTU | * | GLB1_LUCPE | * | HMPA_ALCEU | * | GLB3_LUMTE | * | YBT6_YEAST | |
| 32 | GLB1_LUCPE | * | GLB3_LUMTE | * | FHP_CANNO | * | LGB2_LUPLU | * | GLB1_LUCPE | * |
| 33 | LGB2_LUPLU | * | A1AT_BOMMO | | LGBA_PHAVU | * | LGB1_MEDSA | * | FHP_YEAST | * |
| 34 | GLB1_TYLHE | * | GLB2_LUMTE | * | HBPL_TRETO | * | LGB2_MEDTR | * | PHYB_SOLTU | |
| 35 | GLB4_TYLHE | * | HBPL_PARAD | * | LGB2_MEDTR | * | HBP1_CASGL | * | TGR2_RAT | |
| 36 | LGB1_MEDTR | * | LGBA_PHAVU | * | GLB4_TYLHE | * | HBF1_URECA | * | TAP2_MOUSE | |
| 37 | LGB3_SOYBN | * | GLB2_LUCPE | * | LGB2_LUPLU | * | GLB2_LUMTE | * | TOPB_HUMAN | |
| 38 | GLB1_LUMTE | * | SERA_MANSE | | FHP_YEAST | * | GLB1_CALSO | * | 1A02_GORGO | |
| 39 | LGB2_SOYBN | * | LGB2_SESRO | * | GLB1_TYLHE | * | GLB7_ARTSX | * | MAP3_SCHPO | |
| 40 | LGBA_SOYBN | * | LGB2_MEDTR | * | LGB_PSOTE | * | GLB3_LAMSP | * | FEMB_STAAU | |
| 41 | LGB1_SOYBN | * | GLB7_ARTSX | * | LGB1_MEDTR | * | SAV_SULAC | | RBL_RUTFR | |
| 42 | LGB_PSOTE | * | HMPA_ALCEU | * | GLB1_LUCPE | * | LGB1_MEDTR | * | KUP_ECOLI | |
| 43 | HMPA_ALCEU | * | LGB3_MEDSA | * | LGB2_SOYBN | * | GLB_TUBTU | * | HBPL_PARAD | * |
| 44 | HMPA_ECOLI | * | LGB3_SESRO | * | LGB3_SOYBN | * | GLB4_LUMTE | * | YHB3_YEAST | |
| 45 | LGB1_PEA | * | HMPA_ECOLI | * | LGB3_SESRO | * | RRPL_TSWV1 | | RRP1_IAHTE | |
| 46 | FHP_CANNO | * | GLB_TUBTU | * | IRA2_YEAST | | LGBA_PHAVU | * | RFT1_YEAST | |
| 47 | LGB3_SESRO | * | LGB1_MEDTR | * | LGBA_SOYBN | * | POLG_EMCV | | GLB4_LUMTE | * |
| 48 | FHP_YEAST | * | LGB1_LUPLU | * | LGB1_PEA | * | IC18_PRVIF | | HBPL_TRETO | * |
| 49 | LGB1_LUPLU | * | LGB1_SOYBN | * | LGB2_SESRO | * | HMPA_ALCEU | * | BPS2_DESAM | |
| 50 | LGB2_SESRO | * | GLB_ASCSU | * | LGB1_SOYBN | * | DPOL_VZVD | | VNCA_AAV2 | |
| 51 | LGB1_VICFA | * | LGB2_SOYBN | * | LGB1_LUPLU | * | GLB4_TYLHE | * | YNX5_CAEEL | |
| 52 | GLB1_CALSO | * | GLBH_CAEEL | * | HMPA_VIBPA | * | GLNE_HAEIN | | PRIM_HAEIN | |
| 53 | GLB2_CALSO | * | GLB2_TYLHE | * | MSU1_YEAST | | GSH1_ARATH | | PBPE_BACSU | |
| 54 | HMPA_VIBPA | * | GLB1_LUMTE | * | PHYB_SOLTU | | LGB_PSOTE | * | ICP4_HSV11 | |
| 55 | APA1_RABIT | | LGBA_SOYBN | * | YJK9_YEAST | | YBT6_YEAST | | FHP_CANNO | * |
| 56 | MUP6_MOUSE | | GLB4_TYLHE | * | RRPL_TSWV1 | | NIFK_ANASP | | SCGA_XENLA | |
| 57 | MLRB_CHICK | | LGB1_VICFA | * | USO1_YEAST | | CPSM_HUMAN | | HMPA_ALCEU | * |
| 58 | YFF7_YEAST | | FHP_CANNO | * | PTGA_BACSU | | LGB3_SOYBN | * | EPT1_YEAST | |
| 59 | GLBH_CAEEL | * | GLB2_CALSO | * | COAC_YEAST | | GLB1_LUMTE | * | CLH_DICDI | |
| 60 | PHYB_SOLTU | | HMPA_VIBPA | * | PPOL_MOUSE | | SYK_BACSU | | HBP2_CASGL | * |
| 61 | NIF2_METIV | | FHP_YEAST | * | HD_MOUSE | | PR08_YEAST | | SNQ2_YEAST | |
| 62 | GR78_PHAVU | | GLB1_CALSO | * | YHZ7_YEAST | | LGB1_PEA | * | CSG_METSC | |
| 63 | GLB_ASCSU | * | LGB1_PEA | * | UBA1_HUMAN | | HMPA_VIBPA | * | SLAP_BACTI | |
| 64 | 2AAB_HUMAN | | HS30_ONCTS | | SYK_BACSU | | YY02_HUMAN | | RPA1_SULAC | |
| 65 | SYK_BACSU | | 2AAB_HUMAN | | GLB1_CALSO | * | POLG_COXB5 | | C1R_HUMAN | |
| 66 | YCH2_YEAST | | MYOP_BOVIN | | CAP1_FLATR | | LGB2_SOYBN | * | FL10_CHLRE | |
| erreurs | 10 | | 10 | | 14 | | 21 | | 47 | |

TAB. 5.6 - **Comparaison entre une famille de globines et un ensemble de protéines (66 globines et 1875 non-globines).** *Les «*» marquent les globines. Le nombre d'erreurs représente le nombre de non-globines parmi les* 66 *meilleurs scores.*

# Conclusion

*"Les hommes, n'ayant pas pu guérir la mort, la misère, l'ignorance, ils se sont avisés, pour se rendre heureux, de n'y point penser"*

*Blaise Pascal.*

Lors de ce travail sur la comparaison de séquences biologiques j'ai été amené à considérer des informations autres que la stricte séquence primaire. Le nouvel algorithme SWM, Smith-Waterman avec Motifs, permet de faire intervenir l'information d'un motif au sens PROSITE, et de favoriser les alignements mettant en correspondance deux réalisations d'un même motif. La programmation dynamique classique a été modifiée afin de prendre en compte la non-homogénéité des séquences, modélisée ici par la présence/absence de motifs. La méthode consiste à réaliser un alignement par programmation dynamique lettre à lettre au sens de Smith & Waterman, en attribuant un bonus à tout alignement mettant en correspondance le même motif sur les deux séquences. La pondération des motifs permet de raccorder deux zones éloignées de similarité. Lors de data-bank scanning, le fait de raccorder deux zones de similarité permet de classer les séquences différemment et de faire ressortir des liens entre séquences qui seraient passés inaperçus sans ce raccordement.

Les résultats de l'application de ce nouvel algorithme sont prometteurs et nous poussent à persévérer dans cette direction. Il est important de noter que la prise en compte d'informations extérieures n'a pas fondamentalement changé l'aspect initial de la programmation dynamique appliquée à l'alignement de séquences. Reste un point important à traiter : comment choisir un bon coefficient pertinent du point de vue de la biologie ? Un tel travail doit faire intervenir les quatre disciplines concernées à savoir la biologie, l'informatique, la statistique et la théorie de l'information.

Cet algorithme pourrait être généralisé dans le cas d'occurrences non exactes de motifs, et plus génériquement dans le cas de la prise en compte non pas d'une information booléenne comme la présence/absence d'un motif, mais d'une information continue comme pourraient l'être d'autres informations physico-chimiques ou une densité de probabilité.

L'algorithme développé pourrait aussi être utilisé dans un algorithme d'alignement multiple où on chercherait d'abord à aligner les zones fortement informatives, comme les zones réalisant des motifs. Cependant, les problèmes classiques de l'alignement multiple reposant sur des méthodes d'alignement par paire, ne seront pas résolus, à savoir l'ordre d'agrégation des séquences, et l'alignement lui-même entre les alignements. Il est à noter que le travail d'Abdeddaïm [1] pourrait apporter

beaucoup à la méthode envisagée. L'apport de son travail est de préserver des *blocs* lors de l'alignement multiple: les occurrences de *facteurs communs* aux séquences seront alignées en premier, l'alignement permettant de relier ces régions bien conservées n'ayant lieu qu'après. Si on remplace les régions bien conservées par les occurrences d'un même motif sur différentes séquences, l'algorithme sera capable d'aligner une famille de séquences même si de très longues insertions/délétions sont apparues entre ces motifs.

On ne peut pas séparer complètement la significativité du score du problème précédent. Lorsqu'on replace le score SW obtenu dans un échantillon de scores aléatoires, on prend en compte un autre type d'information, qui est entre autre la distribution en acides aminés et les longueurs des deux séquences. L'intérêt majeur du Z-score réside dans le fait qu'il permet de s'affranchir, au moins en partie, des dépendances en longueurs et en compositions en acides aminés.

Lors de l'alignement multiple, on cherche à agréger des séquences de longueurs différentes. Le score étant dépendant des longueurs, on doit prendre en compte la significativité du score, et non pas le score. Le Z-score semble être une bonne évaluation de cette significativité, et donner de très bons résultats lors de la classification de protéines surtout lorsqu'une méthode souple comme celle de la classification pyramidale est utilisée [9]. Au sein d'une famille de protéines, la classification hierarchique correspond à partitionner la famille. Mais les séquences sont composées de plusieurs fragments ayant des fonctions différentes : une séquence peut contenir plusieurs fragments et appartenir simultanément à plusieurs classes. La classification hierarchique ne peut donc rendre compte des liens entre ces séquences contenant plusieurs domaines. Les pyramides permettent de construire sur cette même famille, au lieu d'une partition au sens strict, un *recouvrement*: chaque individu peut appartenir à une ou deux classes. Ce recouvrement donne beaucoup plus de souplesse pour représenter les différents liens biologiques. Partir de la classification pyramidale serait un bon guide pour l'agrégation des séquences. Il n'en reste pas moins vrai que les problèmes de la significativité soulevés lors de la description des méthodes et résultats révèlent que la question est loin d'être totalement résolue.

Sur le plan mathématique la plus grande question est la justification de l'éloignement de la queue de distribution, du modèle de Karlin. Lorsque le Z-score dépasse un certain seuil, les paramètres du modèle de Gumbel ne sont plus valides, mais la loi reste une loi des valeurs extrèmes. Deux hypothèses sont possibles. Il peut s'agir d'un mélange de population. Mais il est vraisemblable que lorsque les séquences sont très similaires, le processus de Monte-Carlo estime mal la moyenne et l'écart-type. Et le Z-score n'est plus estimé de manière satisfaisante. Il semble cependant important de noter le changement de régime autour de la valeur seuil. Comparer le comportement du Z-score sur des ensembles de séquences réelles et simulées a permis de mettre en évidence une surreprésentation des hauts Z-scores dans le cas des séquences réelles. Ces séquences semblent ne pas être des suites aléatoires de lettres, remarque sensée du point de vue biologique.

La formulation de l'algorithme à l'aide de la structure algébrique $(max, +)$ donne un autre éclairage sur le processus d'alignement de séquences. Le score obtenu par programmation dynamique (Needleman & Wunsch) entre deux séquences $A$ et $B$ de longueurs $l_A$ et $l_B$, coïncide avec la dernière composante d'un vecteur de $\mathbb{R}_{max}^{l_B+1}$ dont le calcul réside dans $l_A$ multiplications d'un vecteur par des matrices carrées de dimension $(l_B + 1)$. Il y a autant de matrices carrées qu'il y a de lettres dans l'alphabet considéré. On montre de plus que l'ensemble des vecteurs qu'on peut construire par multiplications successives de matrices est projectivement fini. Il en résulte que, pour une séquence $B$, un automate peut être construit permettant, après construction, le calcul du score NW entre

les séquences $B$ et $A$, $A$ quelconque, en temps linéaire. Pour l'algorithme de Smith & Waterman un résultat analogue est donné.

Certes, dans la version actuelle cet algorithme n'est pas directement utilisable à cause de la taille de l'automate construit. Cependant, des implémentations différentes pourraient le rendre accessible. La première amélioration pourrait être la construction de l'automate non pas en amont du parcours de la banque de données, mais *au cours* du scanning de la banque. On construirait l'automate au fur et à mesure que l'algorithme a besoin d'un nouvel état. Plus généralement, on peut envisager de découper la séquence requête en petits mots, pour lesquels on construirait les automates, et à partir des résultats obtenus construire des approximations des occurrences de la séquence requête au sens SW.

Le travail présenté ici est à la frontière de plusieurs disciplines et les résultats obtenus aident les biologistes puisqu'ils sont amenés à considérer des alignements qui sont statistiquement significatifs. Même si la significativité statistique n'implique pas la significativité biologique, elle reste un très bon guide. L'étude sur le Z-score et son application à grande échelle sur des génomes entièrement séquencés a permis aux biologistes d'obtenir des résultats importants sur la quantification du phénomène de la duplication. Ces résultats sont en train d'être rédigés, mais on peut d'ores et déjà citer les travaux de Piotr Slonimski [114, 115].

Il est clair qu'il ne s'agit pas de résultats finaux. Nous espérons que l'avenir nous permettra de continuer ces travaux et de contribuer aux avancées de la recherche dans ces différentes disciplines qui, toutes, participent à l'un des domaines les plus productifs du moment.

# Annexe A

# Biologie moléculaire : les principes

*Nous ne pouvons pas entrer ici dans la complexité biologique, mais nous allons essayer de résumer l'essentiel de la biologie moléculaire pour pouvoir aborder avec sérénité les problèmes d'alignements de séquences.*

*Comprendre le langage silencieux mais élégant des cellules vivantes est la quête de la biologie moléculaire moderne. De l'alphabet de seulement 4 lettres, représentant les unités de bases de l'ADN (Acide DésoxyriboNucléique), émerge une syntaxe des processus de la vie, dont la plus complexe expression est l'homme. Le déchiffrage de l'information génétique et l'utilisation de cet alphabet pour former de nouvelles "phrases" sont deux points centraux de la biologie moléculaire. Le défi est de trouver de nouvelles approches pour manipuler un tel volume de données, et de fournir aux chercheurs des outils d'analyse et de calcul, pour avancer dans la compréhension de notre héritage génétique et de son rôle dans la santé et la maladie.*

## A.1 Séquences biologiques

### A.1.1 L'ADN

On peut considérer que les séquences biologiques se classent en deux ensembles : les séquences d'ADN (Acide DésoxyriboNucléique) et les protéines. L'information génétique est stockée dans les séquence d'ADN, alors que les protéines forment les éléments actifs dans l'organisme. On estime que le nombre de protéines dans l'organisme humain est de l'ordre de 100 000.

La molécule d'ADN est composée de l'union de deux brins, enroulés en hélice droite. Chaque brin est constitué par une longue chaîne de **nucléotides**, accrochés les uns aux autres par une liaison chimique spécifique qui donne un sens à la chaîne d'ADN. Le squelette de chacun des deux brins est formé par une succession de molécules de *désoxyribose* reliées entre elles. Chaque molécule comporte une base. Les bases sont au nombres de 4 : *Adénine*, *Guanine*, *Cytosine* et *Thymine*. Les deux chaînes d'ADN s'associent entre elles au niveau de leurs bases. Ces liaisons ne peuvent se faire qu'entre *Adénine* et *Thymine*, ou entre *Guanine* et *Cytosine*. On dit que la Thymine et l'Adénine d'une part, et la Cytosine et la Guanine d'autre part, sont *complémentaires* (Cf. Tableau A.1).

Les liaisons entre les bases sont très faibles, et il suffit de peu d'énergie pour les briser, et ainsi séparer les deux brins d'ADN. Chaque brin de la double hélice possède deux extrémités différenciées appelées $5'$ et $3'$; ce qui donne un sens à la suite des nucléotides. Les orientations des deux brins de la double hélice sont opposées.

La taille des brins d'ADN varie d'un organisme à l'autre. Pour exemple, l'ADN des cellules humaines contient environ 3.5 milliards de nucléotides, alors que chez les bactéries l'ADN est formé

de plusieurs millions de nucléotides. Chez les *eucaryotes*[1], c'est-à-dire les organismes possédant un noyau, la presque totalité de l'ADN est contenu dans le noyau. La longueur des brins d'ADN pose le problème du séquençage des brins. En effet, le séquençage ne peut se faire que sur des fragments relativement courts, et l'ADN doit être au préalable prédécoupé. Des *enzymes de restriction* permettent de découper l'ADN en des endroits bien particuliers. En utilisant deux enzymes de restrictions différentes, on peut reconstituer le brin d'ADN en entier, puisque les coupures n'ont pas lieu aux mêmes endroits.

Puisque l'information génétique est portée par l'ADN, deux questions se posent.

- Comment l'information génétique est passée aux deux cellules, lorsqu'une cellule se divise? L'ADN de la cellule mère doit se retrouver identique dans chacune des cellules filles. La structure de la double hélice permet de comprendre comment se fait la duplication de l'information génétique.
- Comment s'exprime l'information génétique ? Quel est le lien etre l'ADN et les protéines ? Par quel mécanisme les protéines sont-elles synthétisées ? C'est le problème de la traduction de l'information génétique ou de l'expression des gènes.

L'information génétique est stockée sur l'ADN, mais la partie de l'ADN codant pour les protéines ne représente qu'une petite fraction de l'ADN total : de 3 à 5 % chez l'homme. Le segment de l'ADN qui code pour une protéine correspond à la notion de gène. Mais chose curieuse, dans un gène, il y a des régions qui ne codent pas. On les appelle les *introns*, à opposer aux parties codantes : *exons*.

| Bases | Symboles | Compléments | types |
|---|---|---|---|
| Adénine | A | T/U | purine |
| Cytosine | C | G | pyrimidine |
| Guanine | G | C | purine |
| Thymine/Uracile | T/U | A | pyrimidine |

TAB. A.1 - **Nomenclature des Nucléotides :** *Les acides nucléiques diffèrent selon le type de la séquence biologique. Pour les chaînes d'ARN, l'Uracile remplace la Thymine.*

### A.1.2 L'ARN - La transcription

Une autre classe très importante de macromolécules joue un rôle primordial dans l'expression de l'information génétique. Il s'agit de l'**ARN** ou *acide RiboNucléique*. l'ARN a une structure très semblable à celle de l'ADN. Les composants de la chaîne ne sont plus des bases formées sur le sucre *Désoxyribose* mais sur un autre sucre appelé *ribose*, et, la Thymine est remplacée par l'Uracile.

Il y a trois classes d'ARN :

- Les ARN ribosomaux : ces molécules constituent le Ribosome, complexe effectuant la synthèse des protéines.
- Les ARN messagers (ARNm) : ces molécules sont la transcription des gènes, et porte le message génétique utile, c'est-à-dire que l'information génétique codée par les introns a été supprimée.

1. Les eucaryotes sont à opposer aux *procaryotes*, qui sont des organismes unicellulaires n'ayant pas de noyau.

| Bases | Symboles | Signification | Compléments |
|---|---|---|---|
| Adénine | A | A | T |
| Cytosine | C | C | G |
| Guanine | G | G | C |
| Thymine/Uracile | T/U | T | A |
| | M | A ou C | K |
| | R | A ou G | Y |
| | W | A ou T | W |
| | S | C ou G | S |
| | Y | C ou T | R |
| | K | G ou T | M |
| | V | A ou C ou G | B |
| | H | A ou C ou T | D |
| | D | A ou G ou T | H |
| | B | C ou G ou T | V |
| | N | A ou G ou C ou T | N |

TAB. A.2 - **Nomenclature étendue des Nucléotides**

- Les ARN de transfert (ARNt) : Ils sont très nombreux dans chaque cellule, et chaque ARN de tranfert est spécifique d'un seul acide aminé. Les 20 acides aminés constituent les éléments de bases des protéines. L'ARN de transfert fixe à l'aide d'un enzyme appelé *tRNA synthétase* l'acide aminé auquel il est associé.

La **transcription** désigne l'ensemble du processus qui permet la synthèse de ces trois types d'ARN (Cf. figure A.1). La transcription se fait par recopie d'une des chaînes d'ADN. Pour cela il faut savoir où doit commencer la transcription, et donc connaître la localisation de chaque gène. Un enzyme appelé *ARN polymérase* se fixe en amont du gène sur la partie nommée *promoteur du gène*, et permet la séparation des deux brins d'ADN.

Une fois cet ARN synthétisé, il faut enlever les régions correspondant aux introns. Cette étape est appelée *épissage*. Les travaux récents ont montré que l'épissage peut se faire de plusieures manières différentes. Il en résulte qu'un même gène peut coder pour plusieurs protéines.

### A.1.3 Le code génétique

L'ADN est le support de l'information génétique. Mais l'information dont a besoin une cellule, concerne exclusivement l'ordre des acides aminés sur les chaînes polypeptidiques ou protéines, qui sont les éléments actifs dans l'organisme. C'est vraisemblablement la *structure primaire*, c'est-à-dire l'enchaînement des acides aminés, qui conditionne les organisations supérieures de la protéine et son activité biologique. L'information portée par l'ADN réside aussi dans l'ordre d'enchaînement des bases. Comme il n'existe que 4 types de bases et que les acides aminés sont au nombre de 20, il faut au minimum un enchaînement de 3 bases pour définir un acide aminé ($4^3 = 64$ triplets différents possibles). La relation faisant correspondre à chaque triplet un acide aminé porte le nom de **code génétique** (Cf. tableau A.3).

Un suite de trois bases est appelé un **codon**. Puisqu'il y a 64 codons, et qu'il n'y a que 20 acides aminés, le code génétique est dégénéré : plusieurs codons codent pour le même acide aminé. Il existe trois codons «**stop**» : UAA, UAG et UGA. La dégénérescence du code génétique est un avantage

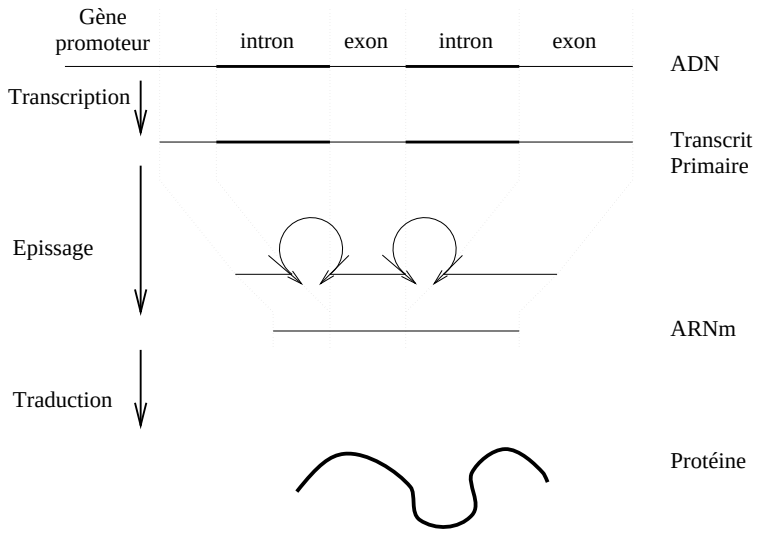



FIG. A.1 - **Transcription.** *Les deux chaînes d'ADN correspondant à un gène se séparent, l'une des chaînes est transcrite en ARN formant le transcrit primaire. Les régions correspondant aux introns sont éliminées par le processus appelé épissage. Les ARN messagers seront ensuite traduit en protéines.*

pour la cellule, puisque une mutation sur un brin d'ADN n'entraîne pas forcément le changement d'un acide aminé dans la protéine. La dégénérescence ne s'est pas établie au hasard, puisque les acides aminés les plus représentés dans les protéines sont ceux qui possèdent le plus de codons.

Les mitochondries possèdent un génome autonome mais très largement insuffisant pour coder toutes les protéines dont elles ont besoin. Le génome des mitochondries a quelques spécificités : entre autres, il est circulaire, et possède un code génétique légèrement différent. Le génome mitochondrial est très compact, puisqu'il n'y a pas de zones non codantes ou **introns**.

Le fait qu'il faille lire trois nucléotides pour connaître l'acide aminé correspondant, implique qu'il y a trois façons de lire un brin d'ADN, déterminée chacune par le nucléotide choisi comme première lettre du codon. Chacune de ces façons détermine une **phase de lecture**. Chacun des deux brins d'ADN peut être lu, et finalement, il existe 6 *phases de lecture* différentes.

## A.2 La synthèse des protéines

La synthèse des protéines est l'ensemble du processus qui, à partir de l'information génétique contenue dans l'ARNm, permet de fabriquer les protéines, éléments actifs associés aux gènes. C'est la phase ultime de l'expression d'un gène. Les ribosomes sont le siège de la synthèse des protéines, et les ARN de tranfert jouent un rôle primordial. Le ribosome lit séquenciellement l'ARN messager, et à chaque codon associe l'acide aminé convenable grâce aux ARN de transfert. Il existe un ARNt pour chacun des acides aminés, et chaque ARNt contient, en une position précise, l'anticodon du codon auquel il est associé, et à une extrémité l'acide aminé pour lequel il code. Le ribosome contient deux sites actifs (appelés P, site peptidique et A, site acide aminé) permettant chacun de lire un codon de l'ARN messager.

La synthèse des protéines des ***eucaryotes***, organismes dont chaque cellule possède un noyau, est différente de celle des ***procaryotes***, organismes sans noyau. Dans ce dernier cas, la synthèse de

| Première Base | Seconde Base | | | | | | | | Troisième Base |
|---|---|---|---|---|---|---|---|---|---|
| | U | | C | | A | | G | | |
| U | UUU | PHE | UCU | SER | UAU | TYR | UGU | CYS | U |
| | UUC | PHE | UCC | SER | UAC | TYR | UGC | CYS | C |
| | UUA | LEU | UCA | SER | UAA | STOP | UGA | STOP | A |
| | UUG | LEU | UCG | SER | UAG | STOP | UGG | TRP | G |
| C | CUU | LEU | CCU | PRO | CAU | HIS | CGU | ARG | U |
| | CUC | LEU | CCC | PRO | CAC | HIS | CGC | ARG | C |
| | CUA | LEU | CCA | PRO | CAA | GLN | CGA | ARG | A |
| | CUG | LEU | CCG | PRO | CAG | GLN | CGG | ARG | G |
| A | AUU | ILE | ACU | THR | AAU | ASN | AGU | SER | U |
| | AUC | ILE | ACC | THR | AAC | ASN | AGC | SER | C |
| | AUA | ILE | ACA | THR | AAA | LYS | AGA | ARG | A |
| | AUG | MET | ACG | THR | AAG | LYS | AGG | ARG | G |
| G | GUU | VAL | GCU | ALA | GAU | ASP | GGU | GLY | U |
| | GUC | VAL | GCC | ALA | GAC | ASP | GGC | GLY | C |
| | GUA | VAL | GCA | ALA | GAA | GLU | GGA | GLY | A |
| | GUG | VAL | GCG | ALA | GAG | GLU | GGG | GLY | G |

TAB. A.3 - **Code génétique**

la protéine peut commencer avant la fin de la transcription. Dans le cas des eucaryotes, le processus se décompose en plusieurs phases (Cf. figure A.2) :

1. L'initiation : L'ARN messager vient se fixer sur le Ribosome au site P par sa partie antérieure, et l'ARN de transfert associé au premier codon vient se fixer au premier codon.

2. L'élongation : Un second ARN de transfert vient se fixer sur le site A du ribosome. Le ribosome se déplace alors d'un codon sur l'ARN messager, et le second acide aminé, toujours attaché à son ARN de tranfert se combinera au premier acide aminé. L'ARN de tranfert du premier codon va être éliminé, et on aura deux acides aminés sur le site peptidique. Le site A est de nouveau libre et cette étape pourra être répétée jusqu'à la fin de la synthèse.

3. La terminaison : Lorsque l'un des codons «**stop**» apparaît sur la chaîne ARN messager, la protéine sera libérée, le ribosome quittera l'ARN messager.

La figure A.3 montre l'ensemble des processus de transcription et de traduction.

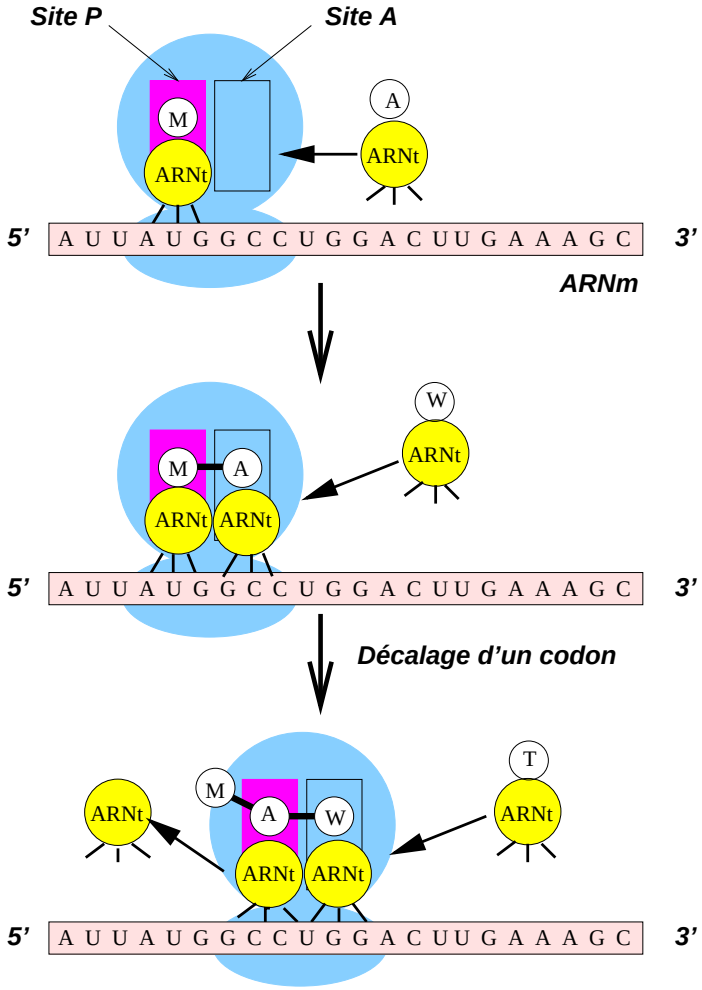



FIG. A.2 - **La transcription**

## A.3 Information génétique et évolution

L'évolution des êtres vivants est la traduction des modifications de l'information génétique. On appelle ces modifications des *mutations*. Lorsque chez l'animal ces mutations ont lieu dans les cellules germinatives, on observera ces mutations chez toute la descendance. Ces mutations sont de trois types :

- ***substitution :*** modification d'un constituant par un autre. C'est le cas le plus fréquent. Les conséquences varient selon la nature de ces changements. Si la substitution a lieu dans une région codante, elle peut être silencieuse, si elle ne modifie pas la signification du codon dans lequel elle apparaît. En effet, puisque le code génétique est dégénéré, plusieurs codons codent pour le même acide aminé et la substitution n'a aucun effet. La substitution peut être néfaste ou non suivant que la protéine garde ou non ses propriétés. Si cette substitution change le codon «**stop**» ou si elle introduit un codon «**stop**» au milieu d'un gène, la protéine synthétisée sera trop longue ou trop courte.

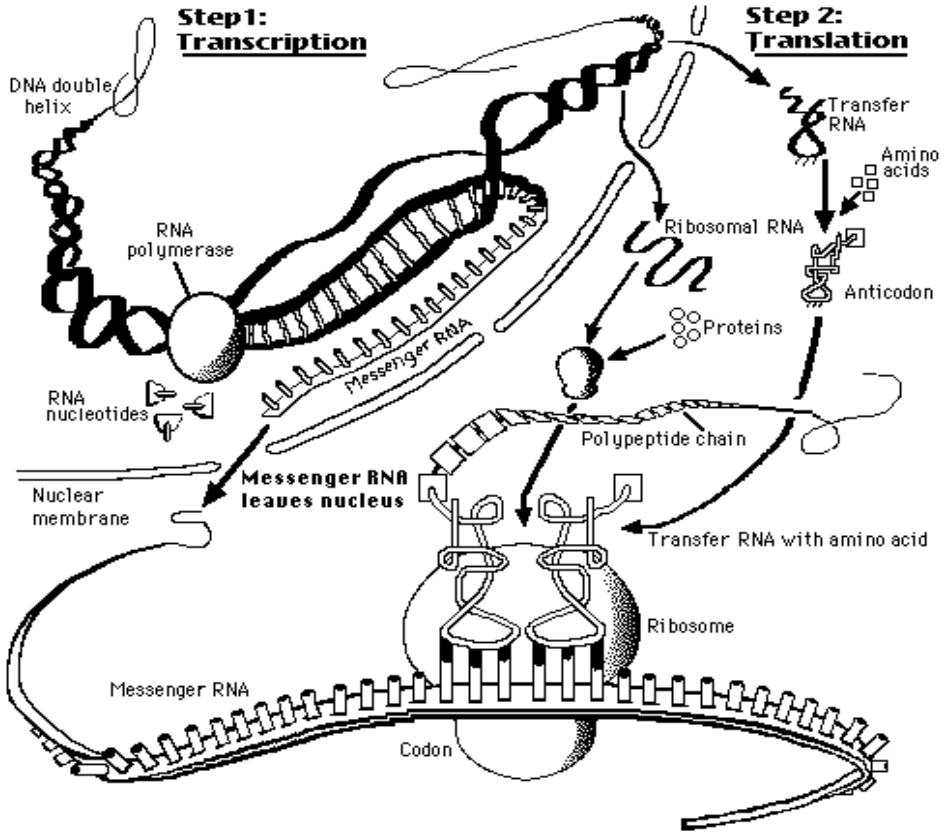



FIG. A.3 - **Synthèse des protéines**

- ***délétion et insertion :*** suppression ou insertion d'une lettre de la séquence. Lorsqu'il s'agit d'une région codante, il y a déplacement de la phase de lecture[2], c'est-à-dire que le début d'un codon a été déplacé d'une ou de deux positions.

- ***duplication :*** une région entière peut être dupliquée. On pourra alors avoir deux gènes identiques. Si la duplication est ancienne, les deux gènes peuvent avoir subi deux mutations différentes, et les deux gènes coderont pour deux protéines différentes.

Chacun de ces mécanismes peut intervenir plusieurs fois, sur la même séquence, pendant une période déterminée.

## A.4 Les protéines

Les protéines sont des macromolécules. Ce sont des chaînes d'acides aminés. Une protéine classique contient entre 200 et 300 acides aminés, mais on trouve des protéines plus petites, et des largement plus longues. La protéine connue la plus grande contient pas moins de 27000 acides aminés.

Les acides aminés sont au nombre de 20. Le tableau A.4 donne la liste des 20 acides, avec leurs abréviations, et leurs caractéristiques élémentaires.

2. On parle en anglais de *frameshift*.

| Symboles (1 lettre) | Symboles (3 lettres) | noms | propriétés |
|---|---|---|---|
| A | Ala | Alanine | petit, apolaire |
| R | Arg | Arginine | gros, polaire |
| N | Asn | Asparagine | petit, polaire |
| D | Asp | Asparatate | petit, polaire |
| C | Cys | Cysteine | petit, apolaire |
| Q | Gln | Glutamine | gros, polaire |
| E | Glu | Glutamate | gros, polaire |
| G | Gly | Glycine | petit, polaire |
| H | His | Histidine | faible polaire |
| I | Ile | Isoleucine | gros, apolaire |
| L | Leu | Leucine | gros, apolaire |
| K | Lys | Lysine | gros, polaire |
| M | Met | Méthionine | gros, apolaire |
| F | Phe | Phenylalanine | gros, apolaire |
| P | Pro | Proline | petit, apolaire |
| S | Ser | Serine | petit, polaire |
| T | Thr | Thréonine | petit, apolaire |
| W | Trp | Tryptophane | faible polaire |
| Y | Tyr | Tyrosine | faible polaire |
| V | Val | Valine | gros, apolaire |
| B | Asp,Asn | | petit, polaire |
| Z | Glu,Gln | | gros, polaire |
| X | | Inconnu | |
| * | | Stop | |

TAB. A.4 - **Nomenclature des Acides Aminés**

Comme toutes les molécules, les protéines adoptent une forme bien particulière dans l'espace. Cette structure *tridimensionnelle* est déterminée en partie par la structure *primaire*, c'est-à-dire la suite des acides aminés, mais aussi par le milieu dans lequel la protéine est plongée (pH, température, autres protéines...).

On appelle structure *secondaire* la conformation locale du squelette de la protéine. Les deux structures secondaires les plus utilisées sont :

- `l'hélice` $\alpha$ est une structure régulière où le squelette s'enroule sur lui-même en formant une hélice. Cette hélice possède 3,6 résidus par tour.

- `Le brin` $\beta$ est une structure beaucoup plus plate, puisque les résidus sont presque complètement étirés. Cette structure n'est stable que lorsqu'elle est présente dans un feuillet $\beta$, où des liaisons faibles apparaissent entre deux brins $\beta$ proche dans l'espace.

La structure *tertiaire* décrit les positions dans l'espace de tous les atomes composant la protéine. La plupart des protéines de grande taille sont composées de domaines qui sont des sous-unités structurales distinctes. Il n'existe pas de définition rigoureuse d'un domaine, mais il est souvent associé à une fonction biologique particulière.

Il est important de noter que la fonction d'une protéine est liée à sa structure tertiaire, plus qu'à sa structure primaire. En effet, en prenant en compte les repliements de la protéine, certains acides aminés éloignés sur la structure primaire se retrouvent très proches dans la structure tertiaire. Ils peuvent alors créer des zones possédant des caractéristiques physico-chimiques particulières appelées *sites actifs*, et responsable de l'activité biologique de la protéine.

La structure de la protéine, qui est fondamentale pour la compréhension de sa fonction, s'obtient soit par des études expérimentales, soit par connaissance de la séquence primaire. L'inférence de la structure tertiaire à partir de la séquence primaire fait l'objet de nombreuses recherches et constitue un pôle important de la bio-informatique.

Pour la comparaison de séquences protéiques, des matrices de substitution ont été introduites, pour prendre en compte des informations physico-chimiques des acides aminés. La famille des matrice BLOSUM est l'une des plus utilisées [3]. La matrice Blosum62 est donnée par la figure A.4.

3. La partie 1.1.4 donne les principes généraux de constructions des matrices du substitution.

```
#  Matrix made by matblas from blosum62.iij
#  * column uses minimum score
#  BLOSUM Clustered Scoring Matrix in 1/2 Bit Units
#  Blocks Database = /data/blocks_5.0/blocks.dat
#  Cluster Percentage: >= 62
#  Entropy =   0.6979, Expected =  -0.5209
   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 -1 -2 -2  0 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -3 -2  0 -2 -1  0 -4
R -1  5  0 -2 -3  1  0 -2  0 -3 -2  2 -1 -3 -2 -1 -1 -3 -2 -3 -1  0 -1 -4
N -2  0  6  1 -3  0  0  0  1 -3 -3  0 -2 -3 -2  1  0 -4 -2 -3  3  0 -1 -4
D -2 -2  1  6 -3  0  2 -1 -1 -3 -4 -1 -3 -3 -1  0 -1 -4 -3 -3  4  1 -1 -4
C  0 -3 -3 -3  9 -3 -4 -3 -3 -1 -1 -3 -1 -2 -3 -1 -1 -2 -2 -1 -3 -3 -2 -4
Q -1  1  0  0 -3  5  2 -2  0 -3 -2  1  0 -3 -1  0 -1 -2 -1 -2  0  3 -1 -4
E -1  0  0  2 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
G  0 -2  0 -1 -3 -2 -2  6 -2 -4 -4 -2 -3 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -4
H -2  0  1 -1 -3  0  0 -2  8 -3 -3 -1 -2 -1 -2 -1 -2 -2  2 -3  0  0 -1 -4
I -1 -3 -3 -3 -1 -3 -3 -4 -3  4  2 -3  1  0 -3 -2 -1 -3 -1  3 -3 -3 -1 -4
L -1 -2 -3 -4 -1 -2 -3 -4 -3  2  4 -2  2  0 -3 -2 -1 -2 -1  1 -4 -3 -1 -4
K -1  2  0 -1 -3  1  1 -2 -1 -3 -2  5 -1 -3 -1  0 -1 -3 -2 -2  0  1 -1 -4
M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1 -3 -1 -1 -4
F -2 -3 -3 -3 -2 -3 -3 -3 -1  0  0 -3  0  6 -4 -2 -2  1  3 -1 -3 -3 -1 -4
P -1 -2 -2 -1 -3 -1 -1 -2 -2 -3 -3 -1 -2 -4  7 -1 -1 -4 -3 -2 -2 -1 -2 -4
S  1 -1  1  0 -1  0  0  0 -1 -2 -2  0 -1 -2 -1  4  1 -3 -2 -2  0  0  0 -4
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -2 -1  1  5 -2 -2  0 -1 -1  0 -4
W -3 -3 -4 -4 -2 -2 -3 -2 -2 -3 -2 -3 -1  1 -4 -3 -2 11  2 -3 -4 -3 -2 -4
Y -2 -2 -2 -3 -2 -1 -2 -3  2 -1 -1 -2 -1  3 -3 -2 -2  2  7 -1 -3 -2 -1 -4
V  0 -3 -3 -3 -1 -2 -2 -3 -3  3  1 -2  1 -1 -2 -2  0 -3 -1  4 -3 -2 -1 -4
B -2 -1  3  4 -3  0  1 -1  0 -3 -4  0 -3 -3 -2  0 -1 -4 -3 -3  4  1 -1 -4
Z -1  0  0  1 -3  3  4 -2  0 -3 -3  1 -1 -3 -1  0 -1 -3 -2 -2  1  4 -1 -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 -4
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1
```

FIG. A.4 - **La matrice de substitution BLOSUM62**

# Annexe B

# Recherche de motifs : pattern matching

## B.1 Pattern matching exact

La recherche d'occurrences exactes d'un mot dans une chaîne de caractères est un problème largement répandu. Soit un motif $m$, de longueur $|m|$ et une chaîne de caractères $C$ de longueur $n \gg |m|$. Le problème est de trouver toutes les positions où le motif $m$ apparaît dans la chaîne $C$.

L'algorithme naïf consiste à comparer le motif $m$ avec tous les facteurs de la chaîne $C$. A la position $j$ de la chaîne $C$, on compare le mot $C[j, j + |m| - 1]$ au motif $m$. On parcourt ce facteur, et on s'arrête dès qu'on arrive à la position $j + |m| - 1$ ou dès que le caractère courant $C[j + i - 1]$ est différent de $m[i]$. On reprend ensuite la recherche en position $j + 1$.
L'algorithme est en $O(|m| \times n)$.
Les algorithmes efficaces prennent en compte à l'étape $j + 1$ des informations provenant des étapes précédentes.

### B.1.1 Algorithme de Morris-Pratt

L'idée de Morris et Pratt [97] est simple: lorsqu'il y a échec à la position $j$, c'est-à-dire lorsque

$$\begin{aligned} m[1, i] &= C[j, j + i - 1] \\ &et \\ m[i + 1] &\neq C[j + i] \end{aligned}$$

on passe non pas à l'étape $j+1$ mais à l'étape $j+k$, avec $k$ judicieusement choisi. La position suivante à traiter est celle où le motif $m$ est susceptible d'être présent. On sait que $m[1, i] = C[j, j + i - 1]$. S'il existe des suffixes de $m[1, i]$ qui sont aussi préfixes de $m$, alors on reprend la recherche aux positions de ces suffixes dans la chaîne $C$. On peut même se restreindre à l'examen de la position contenant le suffixe, qui est aussi préfixe, de taille maximale.

Pour un motif $m$, on peut précalculer la fonction $spm(i)$ qui donne la longueur du suffixe-préfixe maximal, pour chaque position $i$. L'indice $j$ sur la chaîne $C$ n'est jamais décrémenté. L'algorithme devient :

---

$i = 1$;
$j = 1$;
**tant que** $i \leq |m|$ **et** $j \leq n$ **faire**
    **si** $i \geq 1$ **et** $C[j] \neq m[i]$ **alors** $i = 1 + spm(i)$
    **sinon** $i = i + 1$; $j = j + 1$;
**Si** $i > |m|$ **alors** «**Occurrence de** $m$ **débutant à la position** $j - |m|$»

---

La complexité du parcours de la chaîne de caractères est en $O(n)$. On peut montrer que le calcul de la fonction $spm(.)$ peut se faire en $O(|m|)$. La complexité totale de l'algorithme est en $O(n + |m|)$.

### B.1.2 Algorithme de Knuth-Morris-Pratt

Cet algorithme [82] est un raffinement de l'algorithme précédent, qui cherche à éliminer des cas de recherches inutiles. On se place de nouveau dans le cas d'échec où :

$$\begin{aligned} m[1,i] &= C[j, j+i-1] \\ &et \\ m[i+1] &\neq C[j+i] \end{aligned}$$

Le calcul du décalage est maintenant plus fin. On prend en compte la lettre suivante au préfixe-suffixe. Pour qu'un préfixe-suffixe $m[1,i_1]$ de $m[1,i]$ soit retenu, il faut en plus que $m[i_1+1] \neq m[i+1]$.

L'algorithme est le même, sauf qu'on remplace la fonction $spm()$, par la fonction qui renvoie le plus grand suffixe vérifiant la propriété précédente. Le décalage est plus efficace, et l'algorithme plus rapide en pratique, même si dans le pire des cas, la complexité des deux algorithmes est la même. Il a été prouvé en outre que le nombre maximum de comparaisons pour un site $j$ de la chaîne de caractères, dans le cas de l'algorithme Knuth-Morris-Pratt, était borné par une valeur inférieure à celle obtenue dans le cas de l'algorithme Morris-Pratt.

### B.1.3 Algorithme de Boyer-Moore

L'algorithme de Boyer-Moore [21] est considéré comme l'algorithme le plus efficace pour les applications habituelles. L'algorithme parcourt les caractères du motif $m$ de longueur $|m|$, de droite à gauche. En cas d'échec en position $j$ de la chaîne de caractères, l'algorithme utilise cette fois-ci deux fonctions de décalage vers la droite.

Supposons, comme dans la figure B.1, qu'on observe une substitution en $C[j+i-1] = a$ et $m[i] = b$ et qu'on ait $C[j+i, j+|m|-1] = m[i+1, |m|]$ lors de la comparaison pour le site $i$. Un bon décalage à droite du motif peut être :

- d'aligner le segment $C[j+i, j+|m|-1] = m[i+1, |m|]$ avec l'occurrence la plus à droite de ce même sous-mot dans le motif $m$ qui est précédée par une lettre différente de $m[i] = b$ ($1^{ère}$ fonction de décalage, Fig. A),

- d'aligner le plus long suffixe $v$ de $C[j+i, j+|m|-1] = m[i+1, |m|]$ qui est préfixe de $m$ ($1^{ère}$ fonction de décalage, Fig. B),

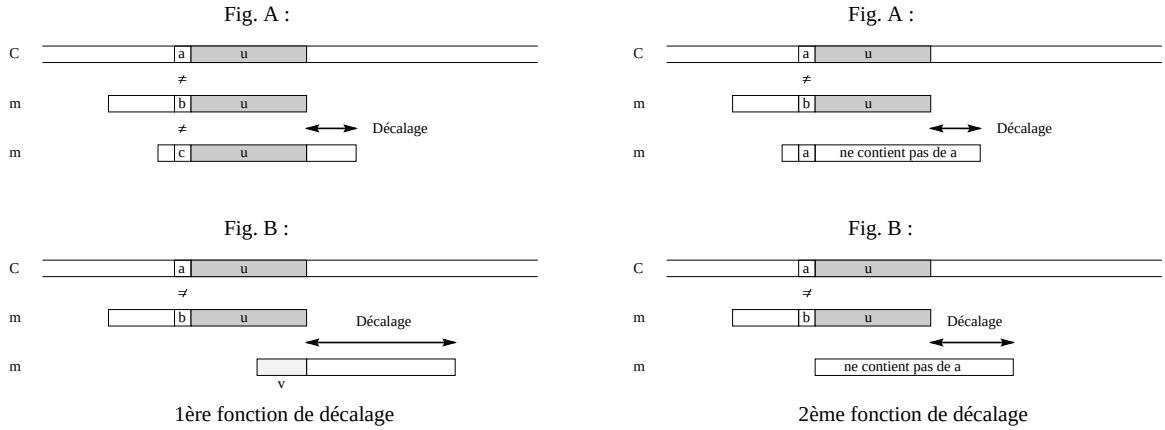



FIG. B.1 - **Les deux fonctions de décalage pour l'algorithme Boyer-Moore.**

- d'aligner le caractère $C[j+i-1]$ avec son occurrence la plus à droite dans $m[1, m-1]$ ($2^{ème}$ fonction de décalage, Fig. A),
- si $C[j+i-1]$ est absent de $m$, d'aligner en position $C[j+i]$ le début du motif $m$ ($2^{ème}$ fonction de décalage, Fig. B).

L'algorithme Boyer-Moore calcule à chaque position le décalage obtenu par chacune des deux fonctions de décalage, puis applique celle qui le maximise.

La construction des deux fonctions de décalage, peut s'effectuer en $O(|m| + |\Sigma|)$, où $|\Sigma|$ est la taille de l'alphabet. L'espace nécessaire pour leur stockage est $O(|m| + |\Sigma|)$. La phase de parcours de la chaîne de caractères $C$ est au pire des cas quadratique. Cependant pour les alphabets de taille importante par rapport à la taille du motif $m$, l'algorithme est performant. Pour les développement autour de cet algorithme, on se reportera à [30][34][39].

## B.2 Pattern matching approché

### B.2.1 Algorithme de Baeza-Yates et Gonnet

**Algorithme «Shift-Or» : cas pattern matching exact.** Baeza-Yates et Gonnet [11] ont donné un algorithme très efficace pour la recherche d'occurrences de mots dans une chaîne de caractères $x$, du moment que le mot $P$ recherché est de petite taille. Cet algorithme est aussi appelé «Shift-And», puisque la formulation de l'algorithme peut être écrite indifféremment avec les opérations logiques «Or» ou «And» Nous avons choisi la formulation «Shift-And» (Cf. éq. B.1).

Définissons $M(i, j)$ un tableau de valeurs binaires :

$$\begin{aligned} M(i,j) &= 1 \;\; si \;\; P[1,i] = x[j-i+1, j] \\ &= 0 \;\; sinon \end{aligned}$$

Clairement $M(n, j) = 1$ si et seulement si une occurrence de $P$ se termine en position $j$ dans la chaîne $x$. L'algorithme calcule d'abord un vecteur $U(\alpha)$ de $n$ valeurs binaires, pour chaque lettre de l'alphabet. La $k^{ème}$ coordonnée de $U(\alpha)$ est mise à 1 pour toutes les positions $k$ où la lettre $\alpha$ est présente dans le mot $P$ ($P[k] = \alpha$).

Définissons la fonction *Bit-Shift*$(j-1)$ qui associe à la colonne $j-1$ de la matrice $M$, le vecteur dont la $1^{ère}$ coordonnée est 1 et dont les autres sont les $n-1$ premières valeurs de la colonne $j-1$ de la matrice $M$.

La matrice $M$ est construite colonne par colonne comme suit :

1. Initialisation : la colonne 1 est initialisée à 0 sauf pour le premier élément qui vaut 1 si $x[1] = P[1]$ et 0 sinon.

2. Boucle : la colonne $j$ est construite à partir de la colonne $j-1$ et de vecteur $U(x[j])$. La colonne $j$ est obtenue par le calcul bit à bit de

$$\textit{Bit-Shift}(j-1) \ \ \textbf{AND} \ \ U(x[j]) \tag{B.1}$$

La valeur de $M(i,j)$ doit être 1 si et seulement si

- les $i-1$ premiers caractères de $P$ sont les mêmes que les $i-1$ caractères de la sous-chaîne de $x$ se terminant en position $j-1$,
- et si $P[i] = x[j]$.

La première condition est vraie si la valeur de $M(i-1, j-1)$ est 1. La seconde condition est vraie si la $i^{ème}$ coordonnée de $U(x[j])$ est 1.

Si la taille du mot recherché est inférieure à la taille d'un mot-machine, chaque vecteur $U(\alpha)$ peut être représenté par un mot-machine. Chaque coordonnée est alors codée par un seul bit. De la même manière, chaque colonne de la matrice $M$ est codée par un mot-machine. Le calcul d'une nouvelle colonne revient à un *seul* calcul sur des mots-machine. Dans ce cas, l'algorithme est très rapide, puisque le calcul d'une colonne se fait en une seule opération sur les mots.

### B.2.2 Algorithme de Tarhio-Ukkonen

Tarhio et Ukkonen ont eu l'idée de généraliser l'algorithme de Boyer-Moore pour l'adapter au cas de la recherche approchée de motif [123].

Le problème dit «k-erreurs» consiste dans la recherche des occurrences approchées d'un mot de longueur $m$ dans un texte de longueur $n$ avec au plus $k$ erreurs. L'algorithme généralisé de Boyer-Moore résout le problème en $O(nk(\frac{k}{|\Sigma|} + \frac{1}{|m|-k}))$ en moyenne, où $|\Sigma|$ est la taille de l'alphabet.

Lorsque $k = 0$, l'algorithme est équivalent à la version de l'algorithme de Boyer-Moore développée par Horspool.

### B.2.3 Algorithme de Wu-Manber dit «Shift-And»

S. Wu et U. Manber [137, 138] ont conçu une méthode fondée sur l'algorithme de Baeza-Yates & Gonnet, pour trouver les occurrences inexactes d'un mot donné. Par occurrences inexactes, nous entendons que le motif apparaît soit de manière exacte, soit avec un petit nombre d'erreurs (substitutions, insertions ou délétions). Leur programme s'appelle *agrep*.

On ne considère que le cas des substitutions. Définissons pour un motif $m$ et un texte $x$ de longueur $m$ et $n$, les matrices $M^k$ de valeurs binaires, où $M^k(i,j)$ vaut 1 si et seulement si les deux mots $m[1,i]$ et $x[j-i+1, j]$ ne diffèrent que de $k$ substitutions au maximum.

$M^0$ est identique à la matrice de l'algorithme de Baeza-Yates & Gonnet. Si $M^k(|m|, j) = 1$, il y a une occurrence inexacte de $m$ qui se termine en $j$ dans le texte $x$, et qui contient au maximum $k$ erreurs.

Pour calculer les matrices $(M^l)_{0 \leq l \leq k}$, on peut calculer toutes les colonnes $j$ de chacune des matrices, puis passer ensuite aux colonnes $j+1$. Toutes les premières colonnes sont initialisées à 0. La colonne $j$ de la matrice $M^k$ est calculée par :

$$M^l(j) = \underbrace{M^{l-1}(j)}_{(a)} \textbf{ OR } \underbrace{[\textit{Bit-Shift}(M^l(j-1)) \textbf{ AND } U(x(j))]}_{(b)} \textbf{ OR } \underbrace{M^{l-1}(j-1)}_{(c)} \qquad \text{(B.2)}$$

Cette équation dit que les $i$ premiers caractères de $m$ correspondent, avec au plus $l$ erreurs, à une sous-chaîne de caractères de $x$ se terminant en $j$ si et seulement si :

- soit les $i$ premiers caractères de $m$ correspondent à une sous-chaîne de caractères de $x$ se terminant en $j$ avec au plus $l-1$ erreurs (terme a),
- soit les $i-1$ premiers caractères de $m$ correspondent à une sous-chaîne de caractères de $x$ se terminant en $j-1$, avec au plus $l$ erreurs, et la paire de caractères $(m[i], x[j])$ est une identité (terme b),
- soit les $i-1$ premiers caractères de $m$ correspondent à une sous-chaîne de caractères de $x$ se terminant en $j-1$ avec au plus $l-1$ erreurs (terme c).

Lorsque le motif est suffisamment petit, pour qu'une colonne d'une matrice $M^l$ puisse tenir sur un petit nombre de mots-machine, l'algorithme est extrêmement rapide. Cependant la complexité de l'algorithme est en $O(k \times |m| \times n)$, en nombre d'opérations sur les bits. Cet algorithme est équivalent à la programmation dynamique.

De la même manière, on peut générer un algorithme du même genre prenant en compte les erreurs du type insertion/délétion. Le calcul d'une colonne est encore une opération logique sur des mots-machine. L'algorithme est équivalent à la programmation dynamique (Cf. section 2.3) à la différence que le nombre maximal d'erreurs autorisées est fixé à l'avance dans le cas de l'algorithme «*Shift-And*».

### B.2.4 Algorithme de Baeza-Yates et Perleberg

Baeza-Yates et Perleberg [12] ont proposé une méthode dont le temps moyen est en $O(n)$, pour des taux d'erreurs petits. Posons $r = \lfloor \frac{|m|}{k+1} \rfloor$. On découpe le motif $m$ en $k+2$ morceaux, dont les $k+1$ premiers ont la longueur $r$.

Si le motif $m$ (de longueur $|m|$) a une occurrence approchée dans le texte $x$, avec au plus $k$ erreurs, alors la sous-chaîne concernée possède au moins un intervalle de longueur $r$ qui est exactement l'une des parties de la partition du motif $m$ (la preuve se fait par l'absurde).

**Algorithme BYP :**

1. Soit $\mathcal{P}$ l'ensemble des $k+1$ premières sous-chaînes de $m$.
2. On construit l'arbre des préfixes de l'ensemble des motifs définis par $\mathcal{P}$. Cette opération se fait en $O(|m|)$ au pire.
3. On utilise l'algorithme de Aho-Corasik pour trouver l'ensemble $\mathcal{I}$ des positions où les motifs de $\mathcal{P}$ apparaissent exactement. La complexité de ce calcul est en $O(n)$ au pire, où $n$ est la longueur du texte à parcourir.
4. Pour chaque $i \in \mathcal{I}$, on utilise un algorithme de pattern matching approché pour localiser les positions de fin des occurrences approchées de $m$ dans la sous-chaîne $x[i-|m|-k, i+|m|+k]$.

Les auteurs ont montré que l'algorithme fonctionne en $O(n(1 + \frac{|m|}{|\Sigma|}))$ en moyenne.

### B.2.5 Complexité

Les développements ont été très nombreux, et ce type de problèmes continue à motiver les chercheurs. Des améliorations apparaissent régulièrement, et le propos n'est pas d'en faire ici la liste exhaustive. Pour indication bibliographique, nous pouvons citer les travaux de Sunday [122, 72], ceux de Stephen [121]. N'oublions pas la synthèse de Gusfield [59] qui fait le lien entre les problèmes de pattern matching et ceux associés à la biologie moléculaire.

Les tableaux B.1 et B.2 synthétisent les complexités des algorithmes de pattern matching avec ou sans erreur. Soit $m$ le motif recherché de longueur $M$. Le texte dans lequel on recherche le motif est de longueur $N$. On pose $M \leq N$.

| Algorithme (sans erreur) | | Complexité en temps | |
|---|---|---|---|
| | | au pire | en moyenne |
| Méthode naïve | | $O(NM)$ | $O(NM)$ |
| Karp-Rabin | [80] | $O(NM)$ | $O(N+M)$ |
| Morris-Pratt | [97] | $O(N+M)$ | délai [a] : $M$ |
| Knuth-Morris-Pratt | [82] | $O(N+M)$ | délai [a] : $1 + c.ln(M)$ |
| Boyer-Moore | [21] | $O(NM)$ | $O(N+rM)$ [b] |
| Boyer-Moore-Horspool | [69] | $O(NM)$ | $O(N+rM)$ [b] |
| Shift-Or | [11] | $O(MN) \vert O(M)$ [c] | |

TAB. B.1 - **Algorithmes exacts de recherche d'un motif dans un texte**

[a] Le délai est le nombre maximal de comparaisons effectuées sur un caractère du texte.

[b] $r$ est le nombre d'occurrences du motif dans le texte. L'algorithme est d'autant plus efficace que l'alphabet est grand. Dans le cas d'un alphabet très grand, le nombre de comparaisons se rapproche de $N/M$.

[c] D'un point de vue purement théorique, cet algorithme est quadratique. il est équivalent à la programmation dynamique. Cependant, si le motif recherché n'est pas trop grand, on peut construire un masque de bits, et l'algorithme devient linéaire.

| Année | Algorithme k-erreurs | | Complexité en temps (en moyenne) | Complexité en espace |
|---|---|---|---|---|
| | Programmation dynamique | | $O(MN)$ | $O(MN)$ |
| 1977 | Hirschberg | [66] | $O(MN)$ | $O(M)$ |
| 1990 | Tarhio-Ukkonen | [123] | $O(M + k\vert\Sigma\vert))$ [a] + $O(Nk(\frac{k}{\vert\Sigma\vert} + \frac{1}{M-k}))$ [b] | $O(k\vert\Sigma\vert)$ |
| 1991 | Wu-Manber | [138] | $O(kN + \vert\Sigma\vert + M)$ | $O(\Sigma.M)$ |
| 1992 | Baeza-Yates-Perleberg | [12] | $O(M + \vert\Sigma\vert)^a + O(N(1 + \frac{M}{\vert\Sigma\vert}))^b$ | $O(M + \vert\Sigma\vert)$ |
| 1994 | Baeza-Yates-Gonnet | [13] | $O(k.M)^a + O(k.N)^b$ | $O(M-k)$ |

TAB. B.2 - **Algorithmes k-erreurs (pattern matching avec erreur) et leur complexité**

[a] Complexité du calcul factorisable : indépendante du texte.

[b] Complexité de l'algorithme de parcours du texte.

# Bibliographie

[1] S. Abdeddaïm. *Comparaison de séquences biologiques.* PhD thesis, Université de Paris VI, 1996.

[2] D. Aldous. *Probability Approximations via the Poisson Clumping Heuristic.* Springer, New York, 1989.

[3] L. Allison and C. N. Yee. Minimum message length encoding and the comparison of macromolecules. *Bulletin of mathematical biology*, 52(3):431–453, 1990.

[4] L. Allison, C. S. Wallace, and C. N. Yee. Finite-state models in the alignment of macro-molecules. *J. Mol. Evol.*, 35:77–89, 1992.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *J. Mol. Biol.*, 215:403–410, 1990.

[6] S. F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.*, 219:555–565, 1991.

[7] R. Arratia, L. Gordon, and L. Goldstein. Two moments suffice for poisson approximations : the Chen-Stein method. *The Annals of Probability*, 17:9–25, 1989.

[8] R. Arratia and M. S. Waterman. A phase transition for the score in matching random sequences allowing deletions. *Ann. appl. Prob.*, 4(1):200–225, 1994.

[9] J.-C. Aude, Y. Diaz-Lazcoz, J.-J. Codani, and J.-L. Risler. Applications of the pyramidal clustering method to biological objects. *Comput. Chem., submitted*, 1998.

[10] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity.* Wiley, 1992.

[11] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, 1992.

[12] R. Baeza-Yates and C. H. Perleberg. Fast and practical approximate string matching. In *Lecture Notes in Computer Science, volume 644 of Combinatorial Pattern Matching ( 3th Annual Symposium, CPM92 )*, pages 185–191. Springer-Verlag, 1992.

[13] R. Baeza-Yates and G. H. Gonnet. Fast string searching with mistakes. *Information and Computation*, 108(2):187–199, 1994.

[14] P. Baldi and Y. Chauvin. Hidden Markov Models of the G-Protein-Coupled Receptor Family. *Journal of computational Biology*, 1(4):311–336, 1994.

[15] A. D. Barbour, L Holst, and S Janson. *Poisson approximation.* Oxford University Press, 1992.

[16] G. J. Barton and M. J. E. Sternberg. Flexible protein sequence patterns: a sensitive method to detect weak structural similarities. *J. Mol. Biol*, 212:389–402, 1990.

[17] G. J. Barton. Protein multiple sequence alignment and flexible pattern matching. *Methods in Enzymology*, 183:403–428, 1990.

[18] G. J. Barton. An efficient algorithm to locate all locally optimal alignments between two sequences allowing for gaps. *CABIOS*, 9(6):729–734, 1993.

[19] R. Bellman and S. Dreyfus. *Applied Dynamic Programming.* Princeton University Press, Princeton NJ, 1966.

[20] P. Bertrand and E. Diday. A Visual Representation of the complexity between an order and a dissimilarity index: The Pyramids. *Computational Statistics Quarterly*, 2(1):31–42, 1985.

[21] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[22] P. Brown, J. Lai, and R. Mercer. Aligning sentences in parallel corpora. In *Proceedings of the 47-th Annual Meeting of the ACL*, 1991.

[23] M. Brown, A. Krogh, I. Saira Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. Technical Report UCSC-CRL-93-32, University of California Santa Cruz, USA., 1993.

[24] J. H. Camin and R. R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19:311–326, 1965.

[25] R. Catizone, G. Russel, and S. Warwick. Deriving translation data from bilingual texts. *First International Workshop on Lexical Acquisition (IJCAI89)*, 1989.

[26] N. Chomsky and M. P. Schützenberger. The algebraic theory of context–free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programing and Formal Languages*, pages 118–161. North Holland, 1963.

[27] L. H. Y. Chen. Poisson approximation for dependent trials. *Ann. Prob.*, 3:534–545, 1975.

[28] O. Chryssaphinou and S. PapasTavridis. A limit theorem for the number of non-overlapping occurences of a pattern in a sequence of independent trials. *J. Appl. Prob*, 25:428–431, 1988.

[29] O. Chryssaphinou and S. PapasTavridis. A limit theorem on the number of overlapping appearances of a pattern in a sequence of independent trials. *Prob. Theory Rel. Fields*, 79:129–143, 1988.

[30] R. Cole, R. Heriharan, U. Zwick, and M. S. Paterson. Tighter lower bounds on the exact complexity of string matching. *SIAM J. Comput.*, 24(1):30–45, 1995.

[31] J. F. Collins, A. F. Coulson, and A. Lyall. The signifiance of protein sequence similarites. *Comput. Appl. Biosci.*, 4:67–71, 1988.

[32] Compugen. *Biocceleratorr manual.* Petcah-Tikva, Israel, 1996.

[33] F. Corpet. Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Research*, 16:10881–10890, 1988.

[34] M. Crochemore and W. Rytter. *Text algorithms.* Oxford University Press, 1994.

[35] J. Devereux. The GCG Sequence Analysis Software Package. *Package, Version 6.0, Genetics Computer Group, Inc., University Research Park, 575 Science Drive, Suite B, Madison, Wisconsin, 53711, USA.*, 1989.

[36] S. R. Eddy. Multiple Alignment Using Hidden Markov Models. In *ISMB*, pages 114–120, 1995.

[37] A. W. F. Edwards and L. Cavalli-Sforza. *Phenetic and Phylogenetic Classification*, chapter Reconstruction of evolutionary trees, pages 67–76. Systemics Association, London, 1964.

[38] Samuel Eilenberg. *Automata, Languages, and Machines,* Volume A. Academic Press, 1974.

[39] N. El Mabrouk. *Recherche approchée de motifs. Application à des séquences biologiques structurées.* PhD thesis, Paris VII, 1996.

[40] J. Felsenstein. Evolutionary trees from DNA sequences : A maximum likelihood approach. *J. Mol. Evol.*, 17:368–376, 1981.

[41] D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol Evol.*, 25:351–360, 1987.

[42] J. W. Fickett. Fast optimal alignment. *Nucleic Acids Research*, 12(1):175–179, 1984.

[43] V. Froidure and J.-E. Pin. Algorithms for computing finite semigroups, in Foundations of Computational Mathematics, F. Cucker and M. Shub eds. *Berlin, Lecture Notes in Computer Science, Springer Verlag*, pages 112–126, 1997.

[44] J. C. Fu. Poisson convergence in reliability of a large linearly connected system as related to coin tossing. *Statistica Sinica*, 3:261–275, 1993.

[45] W. A. Gale and K. W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19:75–90, 1993.

[46] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theoretical computer science*, 64:107–118, 1989.

[47] M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-completeness.* Freeman, 1979.

[48] E. Glémet and J.-J. Codani. LASSAP : a large scale sequence comparison package. *Comp. Appl. BioSci.*, 13(2):137–143, April 1997.

[49] A. P. Godbole. Poisson approximations for runs and patterns of rare events. *Adv. Appl. Prob.*, 23:851–865, 1991.

[50] A. P. Godbole and A. A. Schaffner. Improved Poisson approximations for word pattrens. *Adv. Appl. Prob.*, 25:334–347, 1993.

[51] L. Goldstein and M. S. Waterman. Poisson, compound poisson and process approximations for testing statistical significance in sequence comparisons. *Bulletin of Mathematical Biology*, 54(5):785–812, 1992.

[52] M. Gondran and M. Minoux. *Graphes et Algorithmes.* Eyrolles, 1995.

[53] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Evolution*, 162:705–708, 1982.

[54] P. Green. SWAT: an efficient implementation of the Smith-Waterman or Needleman-Wunsch algorithms. http://bozeman.mbt.washington.edu/phrap.docs/swat.html, 1996.

[55] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis : detection of distantly related proteins. *Proc. Nat. Acad. Sci.*, 84:4355–4358, 1987.

[56] M. Gribskov, R. Luthy, and D. Eisenberg. Profile Analysis. *Methods in Enzymology*, 183:146–159, 1990.

[57] M. Gribskov. Profile Analysis. *Methods in Molecular Biology*, 25:247–266, 1994.

[58] D. Gusfield, K. Balasubramanian, and D. Naor. Parametric optimization of sequence alignment. *Proc. of the Third Annual ACM-SIAM Symposium on Discrete algorithms*, pages 432–439, 1992.

[59] D Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, 1997.

[60] J. J. Hein. A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular Biology and Evolution*, 6:649–668, 1989.

[61] J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bull. Math. Biol.*, 51:597–603, 1989.

[62] J. J. Hein. Unified approach to alignments and phylogenies. *Methods in Enzymology*, 183:626–645, 1990.

[63] J. J. Hein. TreeAlign. *Methods in Molecular Biology, V25: Computer Analysis of sequence Data, Part II*, 25:349–364, 1994.

[64] D. G. Higgins and P. M. Sharp. CLUSTAL : A Package for Performing Multiple Sequence Alignment on a microcomputer. *Gene*, 73:237–244, 1988.

[65] K. Hirano and S. Aki. On number of occurences of success runs of specified lenght in a two-state Markov chain. *Statistica Sinica*, 3:313–320, 1993.

[66] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Comm. A.C.M.*, 18:341–343, 1975.

[67] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *J. of the Association for Computing Machinery*, 24(4):664–675, 1977.

[68] J. L. Holloway and P. Cull. Aligning genomes with inversions and swaps. In *ISMB*, pages 195–202, 1994.

[69] R. N. Horspool. Practical fast searching in strings. *Softw. Pract. Exp.*, 10(6):501–506, 1980.

[70] X. Huang and M. S. Waterman. Dynamic programming algorithms for restriction map comparison. *CABIOS*, 8(5):511–520, 1992.

[71] X. Huang. A context dependent method for comparing sequences. In *5th Annual Symposium CPM*, volume 807 of *LNCS*, pages 54–63. Springer-Verlag, 1994.

[72] A. Hume and D. Sunday. Fast string searching. *Software Practice and Experience*, 21(11):1221–1248, November 1991.

[73] T. Hunkapiller, R. J. Kaiser, B. F. Koop, and L. Hood. Large-scale DNA sequencing. *Current Opinion in Biotechnology*, 2:92–101, 1991.

[74] T. Hunkapiller, R. J. Kaiser, B. F. Koop, and L. Hood. Large-scale and Automated DNA sequence Determination. *Science*, 254:59–67, 1991.

[75] J. W. Hunt and M. D. MacIlroy. An algorithm for differential file comparison. Technical Report TR-41, Computer science, Bell laboratory, 1976.

[76] N. L. Johnson and S. Kotz. *Distribution in statistics : Continuous univariate distributions - 1.* The Houghton Mifflin Series in statistics. 1970.

[77] T. H. Jukes and C. R. Cantor. Evolution of Protein Molecules. *Mammalian Protein Metabolism*, III:21–132, 1969.

[78] S. Karlin and S. F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci.*, 87:2264–2268, 1990.

[79] S. Karlin, A. Dembo, and T. Kawabata. Statistical composition of high-scoring segments from molecular sequences. *Ann. Stat.*, 18:571–581, 1990.

[80] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.

[81] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequence. *J. Mol. Evol*, 16:111–120, 1980.

[82] D. E. Knuth, J. H. Morris, and V. Pratt. Fast pattern matching in string. *SIAM J. Comput.*, 6:323–350, 1977.

[83] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden markov models in computational biology: Application to protein modeling. *J. Mol. Biol.*, 235:1501–1531, 1994.

[84] H. T. Laquer. Asymptotic limits for a two-dimensional recursion. *Studies in applied mathematics*, 64:271–277, 1981.

[85] D. Lavenier. Samba: Systolic accelerators for molecular biological applications. Technical Report 2845, IRISA, 1996.

[86] I. C. Lerman, P. Peter, and J.-L. Risler. Matrices AVL pour la classification et l'alignement de séquences protéiques. Technical Report RR 2466, INRIA, 09-1994 1994.

[87] B. Lewin. *Genes IV*. Cell Press, 1990.

[88] D. J. Lipman, W. J. Wilbur, T. F. Smith, and M. S. Waterman. On the statistical signifiance of nucleic acid similarities. *Nucl. Acids Res.*, 12:215–226, 1984.

[89] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[90] W. J. Masek and M. S. Paterson. A faster algorithm for computing string edit distances. *J. Comput. and System Sci.*, 20:18–31, 1980.

[91] R. J. McEliece. *The theory of information and coding*. Encyclopedia of mathematics and its applications. Addison-Wesley, 1977.

[92] W. Miller and E. W. Myers. Sequence comparison with concave weighting functions. *Bulletin of mathematical Biology*, 50(2):97–120, 1988.

[93] W. Miller and E. W. Myers. Optimal alignments in linear space. *CABIOS*, 4(1):11–17, 1988.

[94] W. Miller, X. Huang, and R. C. Hardison. A space-efficient algorithm for local similarities. *CABIOS*, 6(4):373–381, 1990.

[95] W. Miller and X. Huang. A time-efficient, linear-space local similarity algorithm. *Advances in Applied mathematics*, 12:337–357, 1991.

[96] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2), 1997.

[97] J. H. Morris and V. Pratt. A linear pattern-matching algorithm. Technical Report Report 40, University of California, Berkeley, 1970.

[98] R. Mott. Maximum-likelihood estimation of the statistical distribution of Smith-Watermen Local sequence similarity scores. *Bulletin of Mathematical Biology*, 54(1):59–75, 1992.

[99] M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous comparison of three protein sequences. *Proced. of the Nat. Acad. of sciences*, 82:3073–3077, 1985.

[100] F. Muri. *Comparaison d'algorithmes d'identification de chaînes de Markov cachées et application à la détection de régions homogènes dans les séquences d'ADN.* PhD thesis, Université Paris V, France, 1997.

[101] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[102] C. Neuhauser. A Poisson approximation for sequence comparisons with insertions and deletions. *Ann. Statist.*, 22, 1994.

[103] P. Nicodème. *Alignement avec des familles de séquences protéiques.* PhD thesis, Université de Paris VII, Thèse d' Informatique, 1997.

[104] U. Pape. Algorithm 562 : shortest path lengths. *ACM trans. Math. Software*, 5:450–455, 1980.

[105] W. R. Pearson. Rapid and Sensitive Sequence Comparaison with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

[106] W. R. Pearson. *Methods in Enzymology Vol. 266. Computer methods for macromolecular sequence analysis*, volume 266, chapter Effective protein sequence comparison, pages 227–258. Academic Press, 1996.

[107] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Procedings of the IEEE*, 77(2):257–286, 1989.

[108] N. Saitou and M. Nei. The neighbor-joining method : a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4:406–425, 1987.

[109] S. Sattah and A. Tversky. Additive similarity trees. *Psychometrika*, 42(3):319–345, 1977.

[110] S. Schbath. Compound poisson approximation of word counts in DNA sequences. *ESAIM: Probability and Statistics*, 1:1–16, 1995.

[111] P. H. Sellers. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, 26:787–793, 1974.

[112] P. H. Sellers. The Theory and Computation of evolutionary distances: Pattern recognition. *J. Algorithms*, 1(1):359–373, 1980.

[113] I. Simon. Sequence comparison: some theory and some practice. Technical Report LITP-88-47, Université de Jussieu, Paris, 1988.

[114] P. P. Slonimski, J.-L. Risler, M. O. Mossé, A. Hénaut, Y. Diaz, J.-P. Comet, J-C Aude, E. Glémet, and J.-J. Codani. What can we learn about proteins from complete sequences of microbial genome. *Protein Science*, Vol. 6 Suppl. 1, 1997.

[115] P. P. Slonimski, M. O. Mossé, P. Golik, A. Hénaut, Y. Diaz, J.-L. Risler, J.-P. Comet, J.-C. Aude, A. Wozniak, E. Glémet, and J.-J. Codani. The First Laws of Genomics. *Microbial and Comparative Genomics* in Microbial Genomes II: Sequencing, Functional Analysis and Comparative Genomics (January 31 - February 4), Vol. 3:46, 1998.

[116] T. F. Smith and M. S. Waterman. Comparision of Bio-Sequences. *Advan. Appl. Math.*, 2:482–489, 1981.

[117] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[118] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy.* Freeman, 1973.

[119] E. L. Sonnhammer and D. Kahn. Modular arrangement of proteins as inferred from analysis of homology. *Protein Sci.*, 3(3):482–492, 1994.

[120] R. G. Stanton and D. D. Cowan. Note on a square functional equation. *SIAM Review*, 12(2):277–279, 1970.

[121] G. A. Stephen. String search. Tr-92-gas-01, School of Electronic Engineering Science, University College of N. Wales., October 1992.

[122] D. M. Sunday. A very fast substring search algorithm. *Comm. A.C.M.*, 33:132–142, 1990.

[123] J. Tarhio and E. Ukkonen. Boyer-Moore approach to approximate string matching. In J. R. Gilbert and R. G. Karlsson, editors, *2nd Scandinavian Workshop in Algorithmic Theory, SWAT'90*, volume 447 of *Lecture Notes in Computer Science*, pages 348–359, Bergen, Norway, July 1990. Springer-Verlag.

[124] W. R. Taylor. A flexible method to align large numbers of biological sequences. *J. Mol. Evol.*, 28:161–169, 1988.

[125] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W : improving the sensibility of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[126] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[127] M. Vingron and M. S. Waterman. Sequence alignment and penalty choice : Review of concepts, case studies and implications. *J. Mol. Biol.*, 235:1–12, 1994.

[128] M. Vingron and M. S. Waterman. Rapid and accurate estimates of statistical significance for sequence data base searches. *Proc. Natl. Acad. Sci. USA*, 91:4625–4628, 1994.

[129] P. Vitte. *Plus longue sous-séquence commune et analyse des séquences biologiques.* PhD thesis, Université Aix-Marseille II, 1996.

[130] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Advances in Math.*, 20:367–387, 1976.

[131] M. S. Waterman. Efficient sequence alignment algorithms. *J. Theor. Biol.*, 108:333–337, 1984.

[132] M. S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to tRNA-tRNA comparisons. *J. Mol. Biol.*, 197:723–728, 1987.

[133] M. S. Waterman and M. Vingron. Sequence Comparison Significance and Poisson Approximation. *Statistical Science*, 9(3):367–381, 1994.

[134] W. J. Wilbur and D. J. Lipman. Rapid Similarity Searches of Nucleic Acid and Protein Data Banks. *Proc. Natl. Acad. Sci.*, 80:726–730, 1983.

[135] W. J. Wilbur and D. J. Lipman. The context dependent comparison of biological sequences. *SIAM J. Appl. Math.*, 44:557–567, 1984.

[136] A. Wozniak. Using Video Oriented Instructions to Speed-Up Sequence Comparison. *Comp. Appl. BioSci.*, 13(2):145–150, April 1997.

[137] S. Wu and U. Manber. Agrep : A fast approximate pattern-matching tool. In *Usenix Winter 1992 Technical Conference*, pages 153–162, San Francisco, January 1992.

[138] S. Wu and U. Manber. Fast text searching allowing errors. *Commun. ACM*, 35(10):83–91, October 1992.

[139] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR 94-17, University of Arizona at Tuscon, May 1994.

[140] D. Zajdenweber. Exterme value in Business Interruption insurance. *J. of Risk and Insurance*, 63:95–110, 1996.

[141] S. Gaubert. Performance Evaluation of (max,+) Automata. *IEEE Trans. on Automatic Control*, 40(12), Dec 1995.

# Table des figures

# Liste des tableaux

# Index

# Table des matières

**Title : Dynamic Programming and Biological Sequence Alignments**

**Abstract :**

This thesis is dedicated to methods of biological sequence comparison and alignment, particularly to dynamic programming algorithms.

The Needleman & Wunsch and Smith & Waterman algorithms can be seen as a sequence of elementary operations in algebra (max,+). The use of this algebra allows to imagine a faster equivalent algorithm. One can construct one automata, and afterwards skim through the sequences databank with this automata in linear time. In degenereted cases of dynamic programming the automata is computable in an acceptable time and the algorithm works well.

The existing methods for alignments are based on edition costs computed additionnaly position by position, according to a fixed substitution matrix. It follows from this that a substitution always has the same weight regardless of the position. Nevertheless the biologist favours any similitudes according to his knowledges of structures or functions of the sequences. The laid-out method consists in making other informations operate in the classical dynamic programming algorithm, which can have an effect on the results. The first idea is to take into account the patterns of the databank PROSITE. The method consists in making an alignment with dynamic programming letter by letter as the Smith & Waterman algorithm does, and in assigning a bonus to every path which aligns the same pattern in the two sequences.

The third aspect of this Ph.D concerns the statistical aspects of results of sequence comparisons. It is precious for the biologist to have a guide which allows to class the most significant scores. The most reliable method seems to be the Z-score evaluation, which consists in a Monte-Carlo process simulation. The Z-score is not dépendent of sequence lengths contrary to score. One presents studies over some complete genomes, to validate the Z-score use.

**Keywords :** Sequence comparison, Dynamic programming, Pattern, Algebra $(max, +)$.

**Titre : Programmation Dynamique et Alignements de Séquences Biologiques**

**Résumé :**

Cette thèse est consacrée aux méthodes de comparaison et d'alignement de séquences biologiques et plus particulièrement aux algorithmes reposant sur la programmation dynamique.

Les algorithmes de Needleman & Wunsch et de Smith & Waterman peuvent être vus comme une suite d'opérations élémentaires dans l'algèbre (max,+). L'utilisation de ce formalisme permet de concevoir un algorithme plus rapide strictement équivalent. Il s'agit en effet de construire un automate, puis de parcourir la banque de données grâce à cet automate avec une complexité linéaire. Dans les cas dégénérés de la programmation dynamique, l'automate est calculable en un temps acceptable, et l'algorithme devient performant.

Les méthodes d'alignement existantes sont basées sur des coûts d'édition calculées additivement, position par position, en fonction d'une matrice de substitution fixe. Ceci implique qu'une substitution a toujours le même poids, indépendamment de sa position. Or le biologiste privilégie certaines similitudes en fonction de ses connaissances sur les structures ou fonctions des séquences. La méthode présentée ici, consiste à faire intervenir dans l'algorithme de programmation dynamique d'autres types d'informations pouvant influencer sur le résultat de l'algorithme. La première idée est la prise en compte de motifs. La méthode consiste à réaliser un alignement par programmation dynamique lettre à lettre au sens de Smith & Waterman, en attribuant un bonus supplémentaire à tout chemin mettant en correspondance le même motif sur les deux séquences.

Le troisième aspect de cette thèse porte sur l'aspect statistique des résultats de ces comparaisons. Il est précieux pour le biologiste d'avoir un guide, permettant de déterminer les scores les plus significatifs. La méthode la plus fiable semble être celle du Z-score qui consiste en une simulation de Monte-Carlo. Le Z-score est indépendant des longueurs des séquences, contrairement au score. On présente ici des études sur plusieurs génomes complets, pour valider l'utilisation du Z-score.

**Discipline :** Informatique

**Mots clés :** Comparaison de séquences, Programmation dynamique, Motifs, Algèbre $(max, +)$.

***INRIA Domaine du Voluceau***
***Rocquencourt BP 105***
***78153 Le Chesnay cedex***