Université Paris Cité

Master 2 Bioinformatique - Parcours : Biologie informatique

Du 02 au 13 septembre 2024

ClustalLite : ALIGNEMENT MULTIPLE HEURISTIQUE PAR LA METHODE CLUSTAL

Par Stéphanie Gnanalingam

I) Introduction

1.1) Applications

Clustal [1] est un outil d'alignement multiple de séquences qui permet d'étudier l'homologie entre des séquences biologiques. Lorsqu'il est appliqué à des séquences protéiques, il permet de détecter les positions conservées, ce qui aide à identifier et délimiter des domaines fonctionnels tels que des sites de liaison à des protéines, des promoteurs, des sites actifs, et d'autres motifs, aussi bien dans des protéines apparentées de près que de loin. En utilisant ces alignements, il est possible d'analyser les sites qui subissent une sélection positive ou négative au cours de l'évolution pour des gènes spécifiques. De plus, Clustal peut contribuer à la prédiction de structures tridimensionnelles de protéines en identifiant des régions similaires entre deux séquences, dont l'une a déjà une structure 3D connue.

1.2) Les étapes majeures de l’algorithme

La première étape consiste à effectuer des alignements par paires des séquences en utilisant l'algorithme heuristique de Wilbur et Lipman [1, 2]. Ensuite, un arbre guide est généré à l'aide de la méthode UPGMA pour déterminer l'ordre d'alignement des séquences. Enfin, l'alignement multiple des séquences est réalisé en combinant les algorithmes de Gotoh et de Hirschberg [3].

1.3) Les objectifs

L’objectif du projet *ClustalLite* était de réimplémenter une version simplifiée de l’algorithme de Clustal, en tenant compte des étapes majeures vues précédemment.

II) Matériel et méthodes

2.1) Langage de programmation et environnement

Le programme *ClustalLite* est disponible sur GitHub à l'adresse suivante : https://github.com/gnanalin/ClustalLite

C’est un programme implémenté entièrement en python (v3.11.5) et qui utilise un environnement CONDA (v24.7.1).

2.2) Extraction des données

Le programme prend en entrée un fichier au format FASTA, géré par le module argparse (v1.1) Il est préférable que les identifiants des séquences ne contiennent pas d'espace. Ces séquences sont extraites grâce au module biopython (v1.84).

2.3) Les algorithmes

Les algorithmes ont été implémentés grâce aux modules pandas (v2.2.2) pour la lecture de la matrice BLOSUM62 et numpy (v2.1.1) qui permet de manipuler aisément les matrices. La pénalité de gap est fixée à 8 pour tous les alignements.

2.3.1) Needleman-Wunsch

Pour deux séquences que nous souhaitons aligner, l' algorithme commence par calculer une matrice de score à l'aide de formules prédéfinies. Pour une cellule $F(i,\ j)$ de cette matrice, le score est déterminé par la formule suivante :

$$F(i,\ j)\ =\ max\ (\ F(i-1,\ j-1)\ +\ S(i,\ j)\ \ F(i-1,\ j)\ -\ gap\ \ F(i,\ j-1)\ -\ gap\ )$$

Le score est calculé comme le maximum entre trois valeurs :

- le score provenant de la cellule en diagonale plus le score de substitution des acides aminés donné par la matrice BLOSUM62.
- le score provenant de la cellule au-dessus moins la pénalité de gap.
- le score de la cellule à gauche moins la pénalité de gap.

Simultanément au remplissage de cette matrice de score, une autre matrice est complétée pour conserver le chemin d'origine de chaque score, c'est-à-dire la cellule à partir de laquelle le score de la cellule actuelle a été calculé.

L'étape d'alignement par paires de toutes les séquences du fichier d'entrée a été parallélisée grâce au module joblib (v1.4.2)

2.3.2) UPGMA

La matrice de score issue des alignements par paires est d'abord normalisé pour avoir des distances :

$$matrice\ distances\ =\ 1\ -\ ((matrice\ scores\ -\ minimum)\ /\ maximum\ -\ minimum)$$

Les alignements de score élevés auront donc des distances plus petites.

Dans notre cas, un cluster est défini comme une séquence ou un ensemble de séquences. Une fois les deux clusters $(i,\ j)$ qui ont la distance minimale trouvée, une nouvelle distance des autres clusters k à ce nouveau cluster est calculée :

$$distance\ ((i,\ j),\ k)\ =\ \frac{n_i\ *\ d_{ki} + n_k + d_{kj}}{n_i + n_j}$$

Un dictionnaire python est complété récursivement pour récupérer l'ordre de formation des clusters.

2.3.3) Alignement multiple

Nous appliquons une méthode similaire à l'alignement par paires de Needleman-Wunsch, mais en travaillant avec des clusters (groupes de séquences) plutôt qu'avec des séquences individuelles. Seul le calcul de la diagonale est modifié :

$$diagonale(F(i,\ j))\ =\ \ F(i\ -\ 1,\ j\ -\ 1)\ +\ moyenne(cluster1(i),\ cluster2(j))$$

Ici, $moyenne(cluster1(i),\ cluster2(j))$ représente la moyenne des scores de substitution extraits de la matrice BLOSUM62 pour les positions correspondantes dans les clusters. Si un gap est présent, cette moyenne est ajustée en soustrayant la pénalité de gap.

2.3.4) Complexité temporelle

Lorsque la longueur des séquences double, le temps d'exécution du programme augmente de manière proportionnelle (tableau 1). Cela indique que la complexité temporelle du programme est quadratique.

9 séquences : COMPLEXITÉ EN TEMPS DE O(n²)	
175 acides aminés	14 secondes
355 acides aminés	37 secondes
860 acides aminés	2 minutes 19 secondes

Tableau 1 : Complexité temporelle du programme selon la longueur de 9 séquences.

Résultats

Sortie du programme

L'affichage du terminal se divise en trois parties, correspondant aux sections décrites précédemment. En utilisant le fichier $data/example.fasta$ comme entrée, nous obtenons successivement les figures 1, 2 et 3.

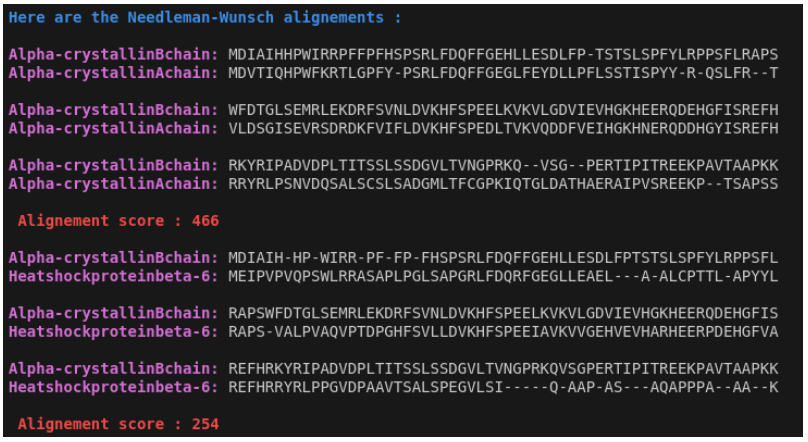

Figure 1 : Une partie des alignements par paires de Needleman-Wunsch avec en bleu, l'introduction de l'étape, en magenta, l'identifiant des séquences et en rouge le score d'alignement.

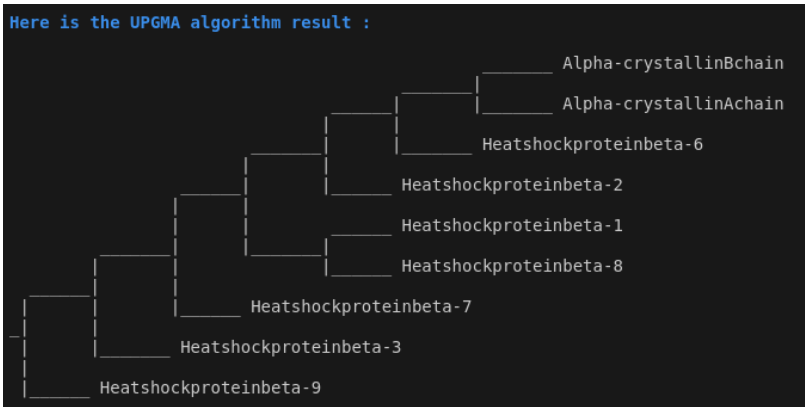

Figure 2 : Arbre guide issu de l'algorithme UPGMA. Cet arbre a été réalisé grâce au module Bio.phylo.

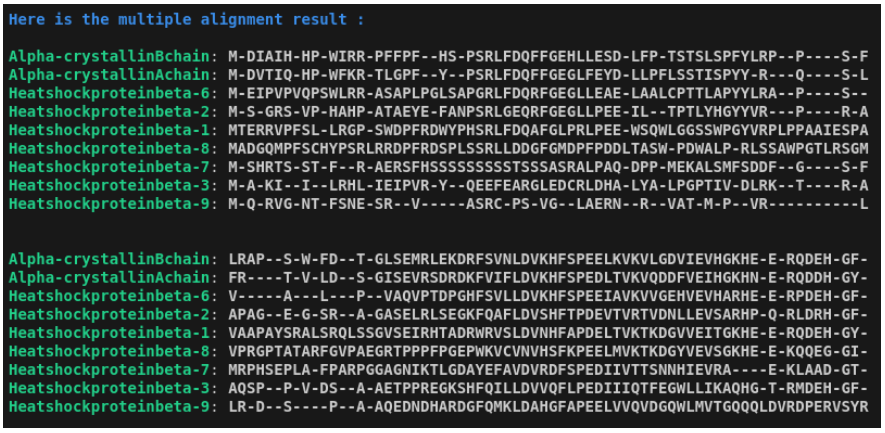

Figure 3 : Alignement multiple avec en bleu, l'introduction de l'étape, en vert, l' identifiant des séquences

Comparaison des résultats avec Clustal Ω

Clustal Ω est la dernière version de Clustal. Les résultats issus de la clusterisation (fig 4) sont légèrement différents. Cependant, cela n'est pas étonnant car Clustal Ω utilise les alignements de Wilbur et Lipman [1, 2] avec une pénalité de gap non constante. De plus, trois méthodes de clusterisation [4] sont appliquées (mBed, k-means et UPGMA).

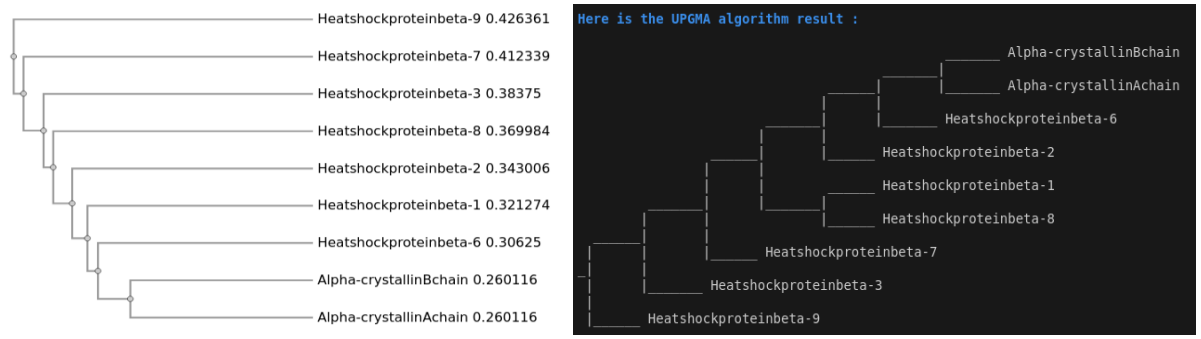

Figure 4 : Algorithme UPGMA sur le fichier de référence $data/example.fasta$ **avec à gauche, le résultat de Clustal Ω et à droite, le résultat de ClustalLite.**

Sachant que l'ordre donné par la clusterisation n'est pas le même, l'alignement multiple (fig 5) est donc différent. De plus, Clustal Ω utilise un modèle de Markov caché, ce qui est plus complexe que notre modèle et qui pourrait donc expliquer les différences.

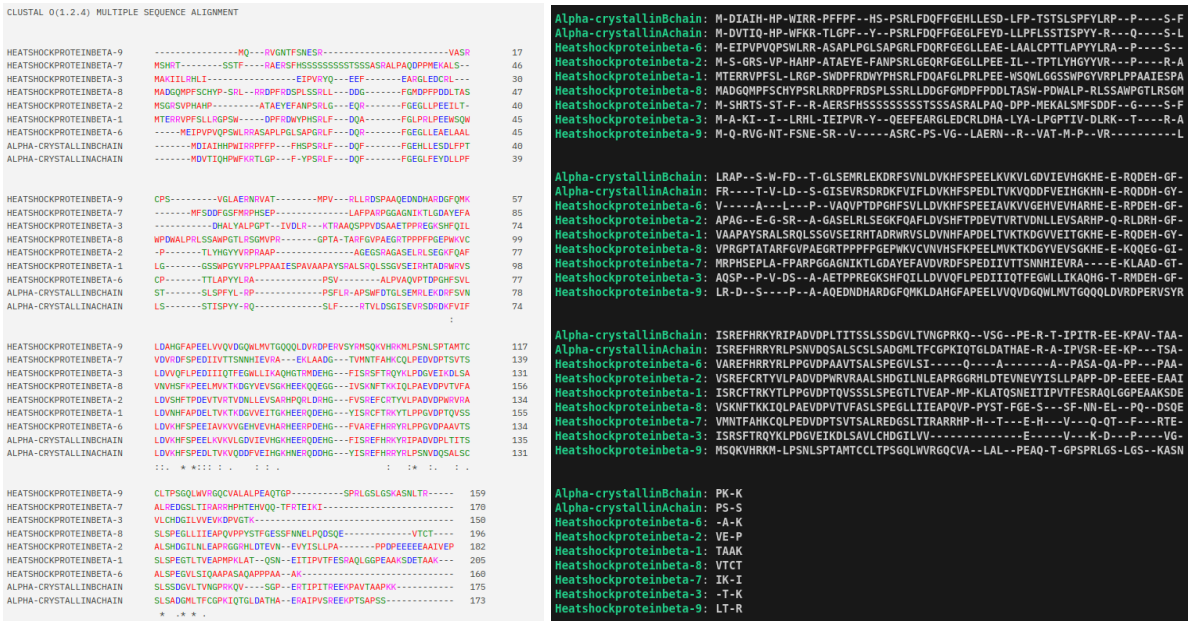

Figure 5 : Alignement multiple sur le fichier de référence $data/example.fasta$ **avec à gauche, le résultat de Clustal Ω et à droite, le résultat de ClustalLite.**

Cependant, il faut souligner que nos alignements ne sont pas aléatoires. Nous retrouvons des positions et zones conservées en termes d'acide aminé et de leurs propriétés physico-chimiques.

Conclusion

L'objectif de ClustalLite était de reproduire l'algorithme de Clustal. Nous avons réussi à implémenter les trois étapes principales : l'alignement par paires, la clusterisation, et l'alignement multiple. Cependant, nos résultats diffèrent de ceux obtenus avec Clustal Ω. Ces différences peuvent s'expliquer par divers paramètres, tels que la pénalité de gap affine, les algorithmes d'alignement par paires, de clusterisation, et d'alignement multiple qui ne sont pas identiques à ceux de Clustal Ω. Malgré ces approximations, nous obtenons des résultats globalement satisfaisants.

Bibliographie

1. Higgins, D. G. & Sharp, P. M. Fast and sensitive multiple sequence alignments on a microcomputer. *Bioinformatics* **5**, 151–153 (1989).

2. Wilbur, W. J. & Lipman, D. J. Rapid similarity searches of nucleic acid and protein data banks. *Proc. Natl. Acad. Sci.* **80**, 726–730 (1983).
3. Hirschberg, D. S. A linear space algorithm for computing maximal common subsequences. *Commun. ACM* **18**, 341–343 (1975).
4. Clustal. *Wikipedia* (2024).