REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : ia6./M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Intelligence Artificielle (IA)**

Approche bio-inspirée pour la résolution du problème d'alignement multiple des séquences ADN

Par :

BENAISSA ABDELBASSET

Soutenu le jj/mm/aaaa devant le jury composé de :

NOM PRENOM	Grade	Président
BERGHIDA Meryem	M.C.B	Rapporteur
NOM PRENOM	Grade	Examinateur

Année universitaire 2020-2021

Remerciements

Tout d'abord, je remercie Dieu de m'avoir donné force et patience, le courage et la volonté durant mes études d'en arriver là et atteindre mes objectifs pour terminer ce travail. Mes profonds remerciements et ma gratitude vont à :

Mme. Meryem BERGHIDA.

D'avoir orienter ce travail dans la bonne direction et pour ses conseils précis et précieux. Je tiens à remercier les membres du jury, nommément, d'avoir accepté de se prononcer sur ce travail dans ces situations exceptionnelles.

Dédicace

Je dédie ce modeste travail

A mes chers parents : ***Abdelkader, Souad*** *pour leur attention, leur soutien et leurs encouragements*

A mes chers frères : ***Ahmed, Boubakeur , Abdelhak*** *et mes sœurs* ***:Safa, Hanane****.*

A mes amies :

LAOUZ HAMED and BASSI Maher and BEKIRI ZeineAbidine

Au famille de ***BEN AISSA*** *et* ***SALHI****.*

Aux gens que je n'oublierai jamais, mes collègues de Groupe d'informatique 2021.

A tous ceux qui m'ont aidé de près ou de loin à mener à bien ce travail.

Résumé

L'alignement multiple de séquences (MSA) constitue une tâche fondamentale pour beaucoup d'applications en bio-informatique. Afin d'évaluer un MSA donné, plusieurs fonctions objectifs ont été définies moyennant des formulations mathématiques. Cependant toutes les méthodes d'alignement proposées dans la littérature reposent sur des approches d'optimisation mono-objectif et fournissent une seule solution potentielle. Dans ce mémoire, nous investiguons l'idée de reconsidérer ce problème sous une vision multi-objectif. La stratégie que nous proposons est un algorithme génétique multi-objectif qui opère sur un ensemble d'alignements dont l'évolution est assurée par des opérateurs génétiques. Après, on améliore les résultats obtenus en utilisant l'algorithme de recuit simulé conduisant à un ensemble d'alignements de meilleure qualité. Les résultats obtenus sont prometteurs et mettent en évidence l'apport de l'optimisation multi-objectif pour la résolution du problème du MSA.

Abstract

Multiple sequence alignment (MSA) is a fundamental task for many applications in bioinformatics. In order to evaluate a given MSA, several objective functions have been defined from the mathematical formulations. However, all the alignment methods proposed in the literature are based on single-objective optimization approaches and offer a single potential solution. In this thesis, we investigate the idea of reconsidering this problem from a multi-objective perspective. The strategy that we propose is a multi-objective genetic algorithm which operates on a set of alignments whose evolution is ensured by genetic operators. Afterwards, the results obtained are improved by using the simulated annealing algorithm leading to a set of better-quality alignments. The results obtained are promising and highlight the contribution of multi-objective optimization for solving the problem of MSA.

Sommaire

Liste des tableaux

Liste des figures

Liste des Algorithmes

Introduction Générale

Introduction générale

La bio-informatique est une interface entre l'informatique et la biologie. Ce champ a plusieurs problèmes ouverts. MSA est le plus connu.

Un alignement est une comparaison de séquences biologiques pour découvrir des sous-séquences similaires. Ceci est motivé par le fait que des protéines ou des gènes avec des sous-séquences similaires sont susceptibles de remplir la même fonction. Une séquence biologique peut être une séquence protéique composée de différents types d'acides aminés ou une séquence d'ADN composée de nucléotides. Il existe 20 types d'acides aminés et 04 types de nucléotides. Si un alignement de séquences implique deux séquences, on parle d'alignement par paires [1,2]. L'objectif principal est de trouver les parties similaires ou étroitement liées. Si l'alignement implique plus de deux séquences, on parle d'alignement de séquences multiples.

L'alignement de séquences multiples est un arrangement simultané de plus de deux séquences ; chacun représenté dans une ligne différente et peut avoir une longueur différente. L'enjeu majeur est de mettre en valeur les résidus identiques dans une même colonne en introduisant le symbole « - », appelé gap. Le MSA est pratique pour l'étude de la fonction moléculaire et de l'évolution. Il permet d'identifier des régions conservées, de définir des relations évolutives, de prédire les structures secondaires ou tertiaires des protéines, la régulation des gènes et la réaction en chaîne par polymérase [3]. L'alignement entre les séquences peut être global ou local. En alignement global, les séquences sont alignées sur toute leur longueur. En revanche, l'alignement local identifie les régions de similitude au sein d'une sous-séquence [4]. Les alignements locaux sont souvent préférables, mais peuvent être plus difficiles en raison du défi supplémentaire d'identifier les régions similaires. Une technique d'alignement global efficace est l'algorithme Needleman – Wunsch [1]. Il est basé sur une programmation dynamique et produit l'alignement optimal par paire. D'autre part, l'algorithme de Smith – Waterman [2] est largement utilisé pour l'alignement local ; il a été inspiré de l'algorithme Needleman – Wunsch.

L'alignement optimal est celui qui décrit le scénario évolutif le plus probable entre toutes les séquences. Il maximise généralement le score mathématique. L'alignement optimal pour deux séquences peut être trouvé en utilisant la programmation dynamique. Mais pour plus de 3 séquences, la complexité augmente considérablement. Pour N séquences de longueur égale à L,

la complexité de calcul est *O (LN).* Ainsi, MSA est un problème NP-difficile [5]. La grande complexité du problème conduit à utiliser des heuristiques pour trouver un alignement quasi optimal avec un coût raisonnable. Plusieurs méthodes ont été développées. Dans ce mémoire, nous proposons deux méthodes approchées bio-inspirées pour résoudre le problème MSA.

La première méthode est l'algorithme génétique AG, qui est un algorithme inspiré de la théorie d'évolution Darwinienne. La deuxième méthode proposée est une combinaison de l'algorithme génétique avec l'algorithme de recuit simulé. Ce dernier, inspiré d'un processus utilisé en métallurgie, améliore la meilleure solution trouvée par l'AG.

Les deux approches sont ensuite testées sur un ensemble d'instances (benchmark BaliBase).

Ce mémoire a été organisé comme suit :

- **Le premier chapitre** : Le premier chapitre présente le problème d'optimisation combinatoire, sa complexité les différentes méthodes d'optimisation et ses classes classiques, ainsi que les algorithmes classiques, les méthodes de résolutions. Nous détaillons ainsi les deux méthodes utilisées dans ce travail, à savoir l'algorithme génétique et le recuit simulé.

- **Le deuxième chapitre** : Dans ce chapitre, nous présentons quelques concepts de base de la bio-informatique en plus du rôle effectif que jouent les techniques informatiques dans la modélisation, l'analyse, la comparaison et la simulation de l'information biologique.

- **Le troisième chapitre** : Ce chapitre est considéré comme le plus important car il explique les méthodes d'alignement basées sur l'analyse de chaînes d'acides aminés ou de protéines, en plus de mettre en évidence les similitudes et les différences.

- **Le quatrième chapitre** : est consacré à la conception de l'algorithme génétique pour résoudre le problème d'alignement, son architecture avec une explication détaillée. Ainsi que la version améliorée de l'AG (amélioration en utilisant le recuit simulé)

- **Le cinquième Chapitre** : est destiné à l'évaluation du résultat des approches proposées en conception. Nous commençons par la description des outils utilisés, ainsi que les expérimentations réalisées et les résultats obtenus.

Chapitre1 : Optimisation combinatoire

1.1 Introduction

Les problèmes d'optimisation consistent à trouver la meilleure solution parmi une gamme de solutions possibles. C'est le problème qu'on rencontre quotidiennement dans la vie réelle. Quand le plus court chemin pour arriver au travail est choisi chaque matin, quand les meubles de la maison sont arrangés de telle façon à ce qu'on gagne plus d'espace, tout en gardant une belle décoration, quand le navigateur GPS aide à trouver une route...etc. L'optimisation est aussi importante dans le domaine commercial. Comment connectez-vous toutes les maisons d'un nouveau quartier avec un nombre minimal de fils électriques ou le nombre minimal de conduites d'égout ? Ou par exemple, quels articles mettez-vous dans le même rayon dans un supermarché ? Quel est le meilleur chemin que vos camions devraient prendre pour aller aux magasins avec un coût minimal (tout en évitant l'embouteillage) ? Où introduire des espaces vides pour justifier (aligner sur les deux côtés) un paragraphe ? ...etc. Dans chacun de ces problèmes, une décision doit être prise. Cette décision influe sur le coût global et la faisabilité de la solution. La façon la plus banale pour résoudre un problème d'optimisation est d'énumérer toutes les solutions faisables correspondantes puis de prendre la meilleure. Cependant, en raison de la nature combinatoire de ces problèmes réels, l'énumération de toutes les solutions possibles est difficile à réaliser, même pour les ordinateurs les plus puissants.

Dans ce chapitre, nous donnons quelques notions de base sur l'optimisation combinatoire, nécessaire pour la réalisation de notre travail.

1.2 Les problèmes d'optimisation combinatoire

Le but d'un problème d'optimisation est de trouver une solution maximisant (resp. Minimisant) une fonction objectif donnée. A chaque problème d'optimisation on peut associer un problème de décision dont le but est de déterminer s'il existe une solution pour laquelle la fonction objective soit supérieure (resp. Inférieure) ou égale à une valeur donnée. La complexité d'un problème d'optimisation est liée à celle du problème de décision qui lui est associé. En particulier, si le problème de décision est NP-complet, alors le problème d'optimisation est dit NP-difficile.[7]

1.2.1. Complexité d'un problème

On entend ici par « complexité d'un problème » une estimation du nombre d'instructions à exécuter pour résoudre les instances de ce problème.[7]

La théorie de la complexité consiste à estimer la difficulté ou la complexité d'une solution algorithmique d'un problème posé de façon mathématique. Elle se concentre sur les problèmes de décision qui posent la question de l'existence d'une solution comme le problème de satisfiabilité booléenne.[6]

1.2.2. Classes de problème

1.2.2.1. La classe P

Contient l'ensemble des problèmes polynomiaux, i.e., pouvant être résolus par un algorithme de complexité polynomiale. Cette classe caractérise l'ensemble des problèmes que l'on peut résoudre « efficacement ».[7]

1.2.2.2. La Classe NP

Contient l'ensemble des problèmes polynomiaux non déterministes, i.e., pouvant être résolus par un algorithme de complexité polynomiale pour une machine non déterministe (que l'on peut voir comme une machine capable d'exécuter en parallèle un nombre fini d'alternatives). Intuitivement, cela signifie que la résolution des problèmes de NP peut nécessiter l'examen d'un grand nombre (éventuellement exponentiel) de cas, mais que l'examen de chaque cas doit pouvoir être fait en temps polynomial. [7]

1.2.2.3. La Classe NP complet

Sont des problèmes combinatoires dans le sens où leur résolution implique l'examen d'un nombre exponentiel de cas. Notons que cette classe des problèmes NP-complets contient un très grand nombre de problèmes. Pour autant, tous les problèmes combinatoires n'appartiennent pas à cette classe. En effet, pour qu'un problème soit NP-complet, il faut qu'il soit dans la classe NP, i.e., que l'examen de chaque cas puisse être réalisé efficacement, par une procédure polynomiale. Si on enlève cette contrainte d'appartenance à la classe NP, on obtient la classe plus générale des problèmes NP-difficiles, contenant l'ensemble des problèmes qui sont « au moins aussi difficiles » que n'importe quel problème de NP, sans nécessairement appartenir à NP.[7]

1.2.2.4. La classe NP-difficile

Un problème de NP est dit NP-difficile, si et seulement s'il existe un problème NP-Complet qu'est réductible à lui en temps polynomial. De cette définition, on conclut que pour montrer qu'un problème d'optimisation est NP-difficile, il suffit de montrer que le problème de décision associé à lui est NP-complet. Ceci explique pourquoi, lors de l'étude d'un nouveau problème, on commence par chercher à classer ce problème. Si l'on parvient à montrer qu'il est polynomial, le problème sera résolu. Si par contre, on parvient à montrer qu'il est NP-complet, la recherche d'un algorithme exact pour résoudre un tel problème ne sera pas de première priorité, et il sera approprié de se concentrer sur des méthodes heuristiques que la plupart des spécialistes de l'optimisation combinatoire ont orienté leurs recherches pour les développer. Une méthode heuristique est souvent définie comme une procédure exploitant au mieux la structure du problème considéré, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible.[8]

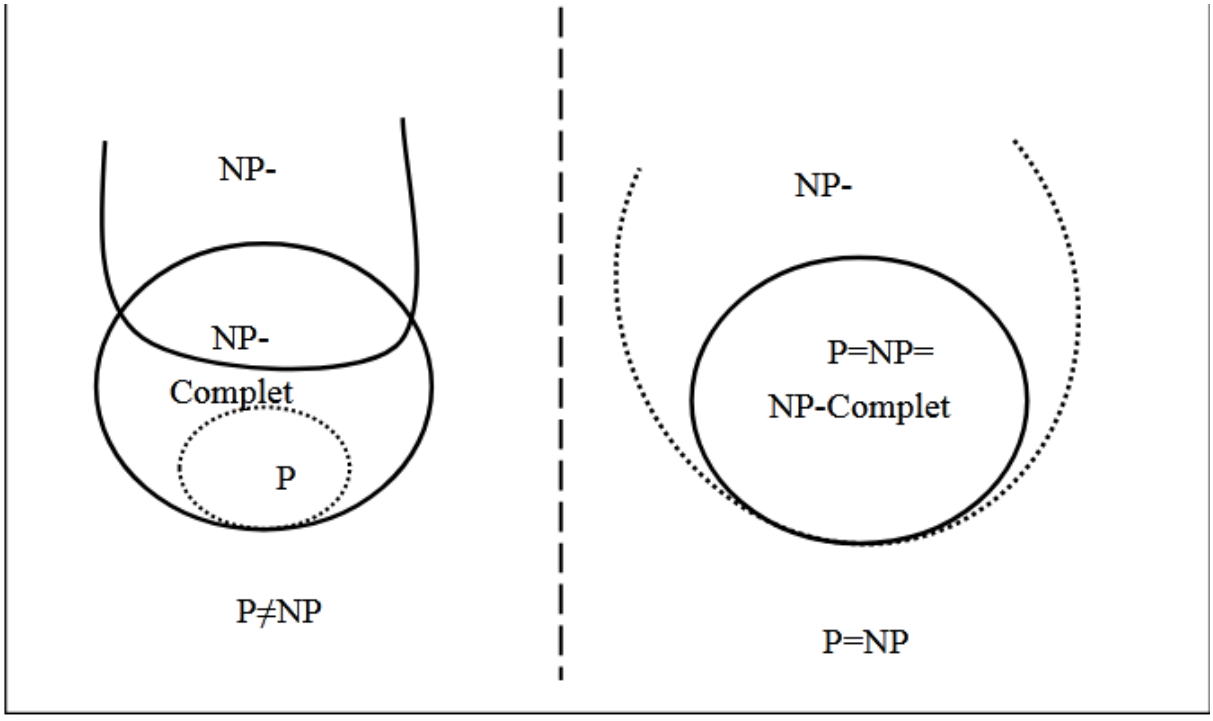

FIGURE 0.1classe P, NP, NP-complet, NP-difficile.

1.3. Notions

1.3.1. Voisinage

Soit une solution s, on dit que s* est une solution voisine de s, si on peut obtenir s* en modifiant légèrement s. On dit aussi qu'on peut passer de s à s* en effectuant un mouvement. Le voisinage V(s) de s est l'ensemble des solutions voisines de s. La fonction V : S →S 2, ∀s ∈ S, V(s)⊂ S est associe à chaque point de S un sous-ensemble de S. [6]

1.3.2. Optimum global :

Une solution est un optimum global à un problème d'optimisation s'il n'existe pas d'autres solutions de meilleure qualité. La solution s* ∈ S est un optimum global si et seulement si : [6]

$$\begin{cases} f(s*) \leq f(s) \text{ Dans le cas d'un problème de minimisation} \\ f(s*) \geq f(s) \text{ Dans le cas d'un problème de maximisation} \end{cases}$$

1.3.3. Optimum local

Une solution s ∈ S est un optimum local si et seulement s'il n'existe pas de solution s0 ∈ v(s), dont l'évaluation est de meilleure qualité que s, soit :[6]

$$\forall s_0 \in v(s) \begin{cases} f(s) \leq f(s0) \text{ Dans le cas d'un problème de minimisation} \\ (s) \geq f(s0) \text{ Dans le cas d'un problème de maximisation} \\ \text{Avec V (s) l'ensemble des solutions voisines de s.} \end{cases}$$

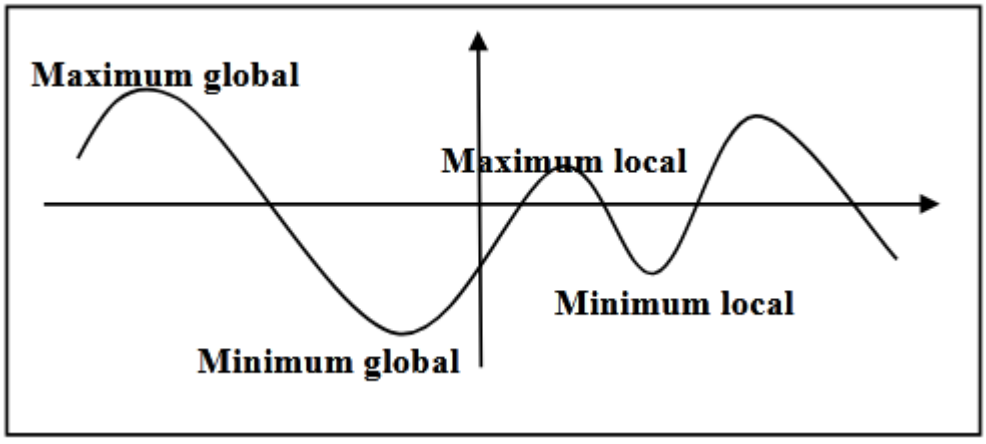

FIGURE 0.2 Optima locaux et optima globaux d'une fonction une variable.

1.3.4. Meilleure solution

Une solution optimale est une solution admissible qui optimise les fonctions objectives.

1.3.5. Diversification, intensification et apprentissage

La diversification (ou exploration, synonyme utilisé presque indifféremment dans la littérature) désigne les processus visant à explorer différentes zones dans l'espace de recherche du problème à optimiser.

Intensification (ou exploitation) vise à parcourir une zone de l'espace de recherche pour trouver la meilleure solution. La mémoire est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi les optima locaux.

Les métas heuristiques progressent de façon itérative, en alternant des phases D'intensification, de diversification et d'apprentissage. L'état de départ est souvent choisi aléatoirement, l'algorithme se déroulant ensuite jusqu'à ce qu'un critère d'arrêt soit atteint. Les notions d'intensification et de diversification sont prépondérantes dans la conception des méta heuristiques, qui doivent atteindre un équilibre délicat entre ces deux dynamiques de recherches. Les deux notions ne sont donc pas contradictoires, mais complémentaires, et il existe de nombreuses stratégies les mêlant.

1.4. Méthodes de résolution des problèmes d'optimisation

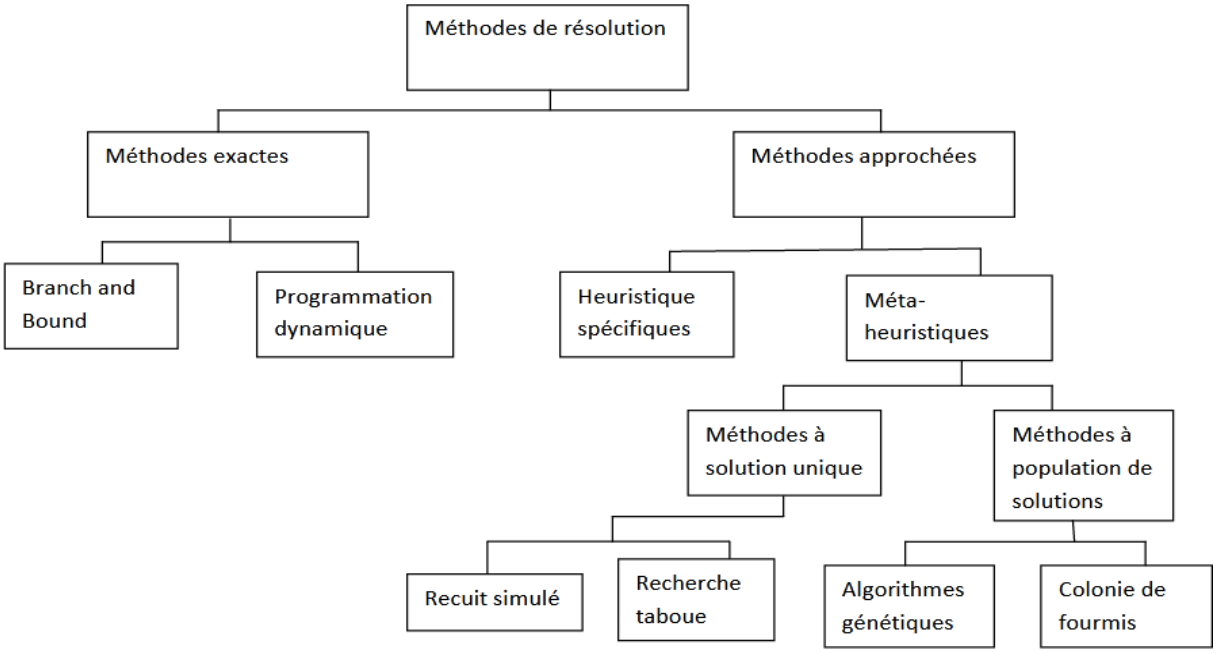

FIGURE 1.03 un schéma montrant les différentes méthodes de résolution en optimisation combinatoire

1.4.1. Les méthodes exactes

Le principe des méthodes exactes consiste à rechercher, souvent de manière implicite, une solution, la meilleure solution ou l'ensemble des solutions d'un problème. L'optimisation exacte concerne toutes les méthodes permettant d'obtenir un résultat dont on sait qu'il est optimal à un problème précis. Cela va des méthodes du simplex aux méthodes de Lagrangien en passant par la programmation dynamique. On peut classer les méthodes exactes en quatre grandes classes. [6]

- La programmation dynamique.
- La programmation linéaire continue ou en nombres entiers.
- La programmation non linéaire avec ou sans contraintes.
- Les méthodes de recherche arborescente (Branch & Bound).

1.4.2. Les méthodes approchées

Dans certaines situations, il est nécessaire de disposer d'une solution de bonne qualité (c'est-à-dire assez proche de l'optimale) dans un contexte de ressources (temps de calcul et/ou mémoire) limitées. Dans ce cas l'optimalité de la solution ne sera pas garantie, ni même l'écart avec la valeur optimale. Cependant, le temps nécessaire pour obtenir cette solution sera beaucoup plus faible et pourra même être fixé (bien évidemment dans ce cas la qualité de la solution obtenue dépendra fortement du temps laissé à l'algorithme pour l'obtenir). Typiquement ce type de méthodes, dites heuristiques est particulièrement utile pour les problèmes nécessitant une solution en temps réel (ou très court) ou pour résoudre des problèmes difficiles sur des instances numériques de grande taille. Elles peuvent aussi être utilisées afin d'initialiser une méthode exacte (Branch & Bound par exemple). Parmi ces méthodes, il faut distinguer les heuristiques ciblées sur un problème particulier et les métas heuristiques plus puissantes et adaptables pour résoudre un grand nombre de problèmes. Cependant une méta heuristique, pour être suffisamment performante sur un problème donné nécessitera une adaptation plus ou moins fine.[9]

1.4.2.1. Les métaheuristiques

Un méta heuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) applicable à un grand nombre de problèmes.[6]Les méta-heuristiques constituent une classe de méthodes qui fournissent des solutions de bonne qualité en temps raisonnable à des problèmes combinatoires réputés difficiles pour lesquels on ne connaît pas de méthode classique plus efficace. On appelle méta-heuristiques (du grec, méta = qui englobe) des méthodes conçues pour échapper aux minima locaux. Le terme méta s'explique aussi par le fait que ces méthodes sont des structures générales dont il faut instancier les composants en fonction du problème par exemple, le voisinage, les solutions de départ ou les critères d'arrêt. Les méta-heuristiques sont généralement des algorithmes stochastiques itératifs qui progressent vers un optimum global, c'est-à-dire l'extremum global d'une fonction en évaluant une certaine fonction objective. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution d'une manière proche des algorithmes d'approximation.

L'intérêt croissant apporté aux méta-heuristiques est tout à fait justifié par le développement des machines avec des capacités calculatoires énormes, ce qui a permis de concevoir des méta-heuristiques de plus en plus complexes qui ont fait preuve d'une certaine efficacité lors de la résolution de plusieurs problèmes à caractère NP-difficile.[6]

1.4.2.2. Les heuristiques

Un algorithme heuristique permet d'identifier au moins une solution réalisable à un problème d'optimisation, mais sans garantir que cette solution soit optimale[11]

Les heuristiques peuvent être classées en deux catégories :

- Méthodes constructives qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue.
- Méthodes de fouilles locales qui démarrent avec une solution initialement complète (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette solution en explorant son voisinage.

1.4.Classification des métaheuristiques :

Plusieurs classifications des méta heuristiques ont été proposé, parmi ces classifications, nous pouvons citer :

1.4.1. Métaheuristiques à base de population ou à trajectoire

Les métas heuristiques peuvent être classées en fonction du nombre de solutions utilisées en même temps. Les méthodes à trajectoire, sont des algorithmes basés sur une solution unique à n'importe quelle étape dans l'algorithme comme les algorithmes de recherche locales tels que TS (Tabu Search) [24], ILS (Iterated Local Search)[25,26] , VNS (Variable Neighborhood Search) [27], HC (Hill Climbing) [28], SA (Simulated Annealing)[29,30], GRASP (Greedy Randomized Adaptive Search Procedure)[31,32]...etc.

Les algorithmes à base de population effectuent la recherche avec plusieurs pointsde départ (solutions initiales) dans un style parallèle comme par exemple : ACO (Ant Colony Optimization algorithm) [33], PSO (Particle Swarm Optimization) [34], les algorithmes évolutionnistes (Les

algorithmes génétiques[35], la programmation génétique [36], les algorithmes émétique [37],les algorithmes d'évolution différentielle [38], la recherche par dispersion[39]...etc.).

1.4.2. Métaheuristiques inspirées ou non inspirées d'un phénomène naturel

Les métas heuristiques inspirées d'un phénomène naturel peuvent se baser selon la source d'inspiration sur :

- **L'intelligence en essaim** comme : PSO, ACO, RFD (River Formation Dynamics) [40], ABC (Arti_cial Bee Colony) [41], BFO (Bacterial Foraging Optimization)[42], FA (Fire_y Algorithm) [43], CS(Cuckoo Search) [44], IWD (Intelligent Water Drops)[44], BA (Bat Algorithm)[43]...etc).

- **Les systèmes biologiques** comme : AG, ACO, CSA (Clonal SelectionAlgorithm) [46] ...etc.

- **La physique ou la chimie** comme : Le recuit simulé (SA), HS (Harmony Search) [47], EO (Extremal Optimization) [48] ...etc. Tandis que la méthode de descente (HC), ou la recherche Tabou (TS), vont dans la seconde classe.

1.4.3. Métaheuristiques avec fonction objectif statique ou dynamique

Les métas heuristiques peuvent également être classées en fonction de la manière d'utilisation de la fonction objectif. Bien que certains algorithmes maintiennent lafonction objectif donnée dans la représentation du problème telle qu'elle, d'autres, comme GLS (Guided Local Search) [49], la modifient lors de la recherche. L'idée derrière cette approche est de s'échapper des minimas locaux en modifiant le paysage de la recherche. En conséquence, lors de la recherche, la fonction objective est modifiée en essayant d'intégrer les informations recueillies au cours du processus de recherche.

1.4.4. Métaheuristiques avec une ou plusieurs structures de voisinage

La plupart des méta heuristiques travaillent sur une structure de voisinage unique. D'autres métas heuristiques, comme la recherche à voisinage variable (VNS), utilisent un ensemble de structures de voisinage qui donne la possibilité de diversifier la recherche et d'explorer mieux l'espace de recherche.

1.4.5. Métaheuristiques avec ou sans mémoire

Une caractéristique très importante pour classer le méta heuristique est l'utilisation de l'historique de recherche. Les algorithmes qui n'utilisent pas la mémoire(historique) effectuent un processus de Markov, car l'information qu'ils utilisent pour déterminer la prochaine action est l'état actuel du processus de recherche. Il existe plusieurs façons pour utiliser la mémoire. Habituellement, nous distinguons entre l'utilisation de la mémoire à long terme et court terme. La première garde généralement la trace de mouvements effectués récemment, des solutions visitée sou, en général, les décisions prises. La seconde est généralement une accumulation de paramètres synthétiques sur la recherche.

1.5.L'algorithme génétique

Les algorithmes génétiques appartiennent aux algorithmes évolutionnaires. Ils s'inspirent de la théorie d'évolution de Darwin. L'algorithme a été proposé par J.Holland 1975, mais sa popularité revient à la parution du livre de D.E.Goldberg 1989 . L'algorithme génère une population de solution à chaque itération en utilisant des individus (solutions) de la population de l'itération précédente et en appliquant sur ses derniers des opérateurs de sélection et de variation. Les opérateurs de sélection permettent à un individu d'être choisi pour la reproduction (croisement). Les opérateurs de variation peuvent être classés en deux catégories opérateurs de croisement et opérateurs de mutation. Les opérateurs de croisement s'appliquent sur chaque deux solutions (parents) sélectionnés de la population et produisent une ou plusieurs nouvelles solutions (enfants). Les opérateurs de mutation, apportent des modifications à une seule solution (individu) et génèrent une nouvelle solution (individu). [13]

FIGURE 0.4 *Le paradigme de la solution*

Il y a trois opérateurs d'évolution dans les algorithmes génétiques :

1.5.1. La sélection

La sélection consiste à choisir les individus les mieux adaptés afin d'avoir une population de solution la plus proche qui converge vers l'optimum global. Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. En règle générale, la probabilité de survie d'un individu sera directement reliée à son efficacité relative au sein de la population.[14]

Il existe plusieurs techniques de sélection. Voici les principes utilisés :

1.5.1.1.Sélection par rang

Cette technique de sélection choisit toujours les individus possédant les meilleurs scores d'adaptation.

1.5.1.2.Probabilité de sélection

Proportionnelle à l'adaptation : Technique de la roulette ou roue de la fortune, pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème.

1.5.1.3.Sélection par tournoi

Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit parmi ces paires l'individu qui a le meilleur score d'adaptation.

1.5.1.4.Sélection uniforme

La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation.

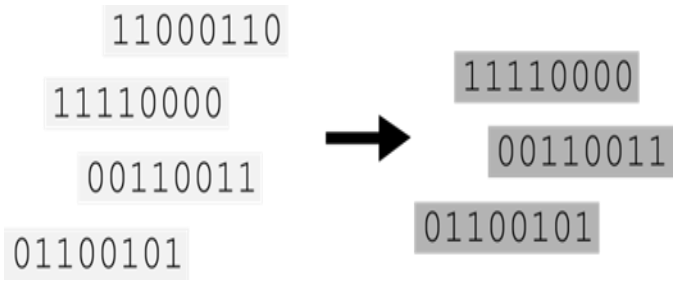

FIGURE 0.5 La sélection.

1.5.2. Le croisement

Le croisement consiste à générer un ou deux nouveaux individus `a partir d'un couple de parents choisi par l'opérateur de sélection. C'est un opérateur qui permet la création de nouveaux individus en combinant les gènes des individus parents. [14]

Il existe deux méthodes de croissement : simple ou double croisement.

• Le croisement simple consiste à fusionner les particularités de deux individus a partir d'un pivot, afin d'obtenir un ou deux enfants

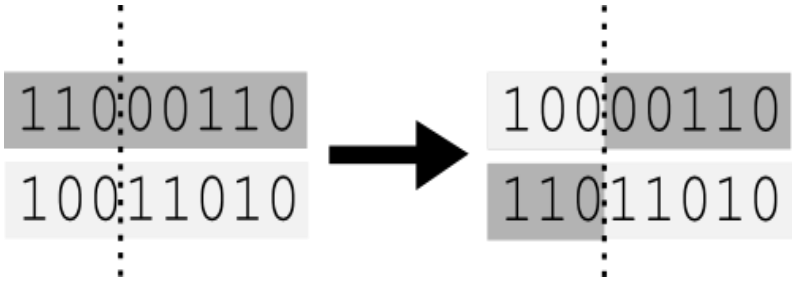

FIGURE 1.6 *Croisement avec un point.[14]*

- Le croisement double repose sur le même principe, sauf qu'il y a deux pivots :

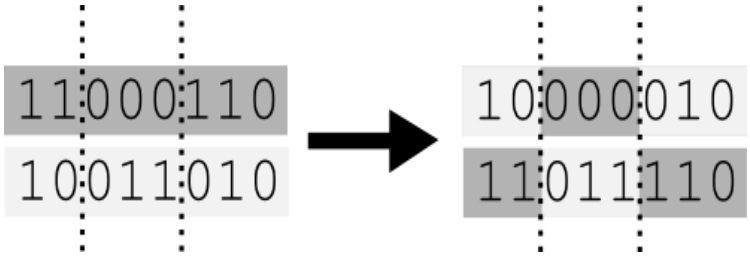

FIGURE 1.7 *Croisement avec deux points.[14]*

1.5.3. La mutation

Une mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation Ou simplement en l'inversion d'un bit (ou de plusieurs bits, mais vu la probabilité de mutation c'est extrêmement rare) se trouvant en un locus bien particulier et lui aussi déterminé de manière aléatoire [9].

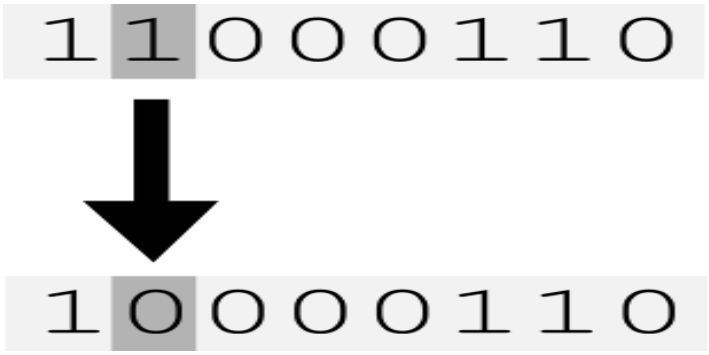

FIGURE 01.8 *La mutation. [14]*

1.6.Recuit simulé

Le Recuit Simulé « Simulated Annealing SA » est inspiré du processus de recuit dans la métallurgie. Dans ce procédé, un matériau naturel est chauffé et refroidi lentement dans des conditions contrôlées pour augmenter la taille des cristaux dans le matériau et de réduire les défauts. Ce ci améliore la résistance et la durabilité du matériau. La chaleur augmente l'énergie des atomes, leur permettant de se déplacer librement, et le programme de refroidissement lent permet qu'une nouvelle configuration à faible énergie à être découverte et exploitée. C'est l'idée

clé qui sous-tend l'algorithme du recuit simulé, qui a été proposé indépendamment par Kirkpatrick et al [29]. Et Cern'y [30].

La méthode est une adaptation de l'algorithme de Metropolis-Hastings, une méthode de Monte Carlo pour générer des états exemples d'un système thermo dynamique, inventée par Rosenbluth et publiée dans un article de Metropolis et al [10]

Chaque configuration d'une solution dans l'espace de recherche constitue une énergie interne différente du système. Le chauffage des systèmes conduit en un assouplissement des critères d'acceptation des échantillons prélevés dans l'espace de recherche. Comme on refroidit le système, les critères d'acceptation des échantillons se rétrécissent pour se concentrer sur l'amélioration de mouvements. Une fois le système refroidi, la configuration représentera un échantillon qui a ou est près d'un optimum global [12].

L'idée est de permettre des modifications locales susceptibles d'augmenter le coût (si cette fonction est à minimiser). Ces modifications vont peut-être permettre de franchir une barrière et aboutir à une solution meilleure. Le processus de recuit simulé est contrôlé par différents paramètres. Les bonnes valeurs pour ces paramètres ne sont pas précisément connues. Le processus doit donc être évalué avant d'être utilisé à l'échelle réelle de façon à ajuster ces divers paramètres. Supposons que le coût d'une configuration Xi soit f(Xi). Une modification locale fait passer à une configuration Xj de coût f(Xj). La décision d'accepter Xj à la place de Xi est prise de la manière suivante :

Soit $\Delta fij=f(Xj)-f(Xi)$;

- Si $\Delta fij \leq 0$, accepter Xj ;
- Si $\Delta fij>0$ accepter Xj avec la probabilité $\exp(-\Delta fij / T)$, ou T est appelée température.

Cette distribution de probabilité est inspirée de la loi de physique statistique de Gibbs-Boltzmann. Dans le contexte de l'informatique, ce choix est connu sous le nom de règle de Metropolis. On voit que, plus la température est élevée, plus la probabilité d'accepter une configuration moins bonne est grande. Par contre, à la fin du processus (lorsque la température

est basse), seules les modifications faisant diminuer le coût seront acceptées. L'algorithme 18 présente le pseudo code de l'algorithme du recuit simulé pour un problème de minimisation.

Algorithme 18 Algorithme de Recuit Simulé

$S_{courante} - CreerSolInitiale()$
$S_{best} - S_{courante}$
pour $i - 1$ à $Nbr_Iteration$ **faire**
  $S_i - CreerSolVoisine(S_{courante})$
  $Temperature_{courante} - CalculerTemp(i, Temperature_{max})$
  si $Cout(S_i \leq Cout(S_{courante}))$ **alors**
    $S_{courante} - S_i$
    si $Cout(S_i) \leq Cout(S_{best}))$ **alors**
      $S_{best} - S_i$
  sinon
    si $exp(\frac{Cout(S_{courante}) - Cout(S_i)}{Cout(S_{courante})}) > Rand()$ **alors**
      $S_{courante} - S_i$

Algorithme1 Algorithme de Recuit simule

1.1. Conclusion

Dans ce chapitre, nous avons passé en revue quelques notions de base de l'optimisation combinatoire à savoir : la complexité d'un problème, les classes de problèmes, les méthodes de résolution des problèmes d'optimisation (heuristiques et métaheuristiques) Et on a détaillé les deux méthodes utilisées dans ce travail : les algorithmes génétiques et le recuit simulé.

.

Chapitre2 :
La Bio-informatique

2.1 Introduction

C'est une recherche originale en informatique, voire en mathématiques/statistiques, suscitée par un problème biologique, qui peut éventuellement conduire à l'acquisition de connaissances en biologie. C'est le cas, par exemple, des recherches menées sur les répétitions (exactes, inexactes, palindromiques) dans les séquences d'ADN et leur compression ; ou de la démonstration théorique par Karlin que les scores d'alignement (sans gaps) des séquences biologiques suivent une distribution dite des valeurs extrêmes — ce qui donnera lieu à l'écriture du programme BLAST ; ou encore de la mise en évidence de mots sur- ou sous-représentés dans les séquences nucléotidiques, qui pose des problèmes statistiques épineux. Les recherches de ce type, qui sont publiées dans des journaux spécialisés, sont le plus souvent inconnues des biologistes .La bio-informatique, ce peut être aussi la mise en œuvre — pas forcément triviale — de méthodes, de concepts ou d'algorithmes éprouvés pour résoudre un problème posé par les biologistes : par exemple, la comparaison de séquences génomiques complètes, ou l'utilisation de la transformée de Fourier pour créer des alignements multiples, ou encore la mise en musique des chaînes de Markov pour repérer les gènes codant les protéines dans les séquences génomiques. Il y a là production, par des informaticiens/mathématiciens/statisticiens, de programmes que les biologistes utiliseront.[15]

2.5.Informatique et biologie moléculaire

2.5.1. La bio-informatique

La bio-informatique est l'étude de l'information biologique. Ce n'est pas simplement l'application à la biologie de l'informatique ; c'est une branche à part entière de la biologie. La bio-informatique actuelle se concentre surtout sur l'étude des séquences d'ADN et sur le repliement des protéines.

2.2.2 biologie moléculaire :

C'est l'étude des molécules porteuses du message héréditaire (ADN, ARN), de leur structure, leur synthèse, leur altération et du contrôle de l'expression des gènes.

2.3 Notions de base en biologie moléculaire

2.3.1 L'ADN : acide désoxyribonucléique

L'ADN, abréviation d'acide désoxyribonucléique, se trouve dans le noyau de la plupart des types de cellules. Il contient les instructions propres à la cellule et détermine comment les traits

d'une personne seront transmis d'une génération à l'autre. Dans le noyau d'une cellule humaine, on compte 23 paires de chromosomes, soit 46 chromosomes en tout. Chaque chromosome est formé de chromatine enroulée qui est composée d'ADN enrobant des protéines appelées histones. Les 23 paires de chromosomes dans le noyau font office de « manuel d'instructions » pour le développement d'un individu. L'ADN détermine si la personne aura les yeux bleu sou bruns ou les cheveux foncés ou blonds.[16]

2.3.2 L'ARN : Acide Ribonucléique

L'ARN est l'abréviation d'acide ribonucléique. Tout comme l'ADN, les molécules d'ARN sont fabriquées dans le noyau de la cellule. Toutefois, contrairement à l'ADN, l'ARN ne se limite pas au noyau. Il peut migrer dans d'autres parties de la cellule. De l'ARN, appelé ARN messager communique le message génétique que l'on retrouve dans l'ADN au reste de la cellule afin de favoriser la synthèse de protéine. La façon dont une séquence de gènes dans [16] L'ADN se traduit par la suite en une protéine correspondante fait l'objet de la section sur la synthèse des protéines. [16]

2.3.3 Protéine

Les protéines sont les macromolécules les plus importantes. Elles sont responsables de presque de toutes les réactions biochimiques qui ont lieu à l'intérieur de la cellule. Les protéines sont de sortes différentes et avec une variété de fonctionnalités. Certaines d'entre elles incluent :

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

Les protéines se composent de chaîne des acides aminés. Chaque acide aminé a une structure constante. Il y a 20 acides aminés. Deux acides aminés peuvent se joindre, avec un " lien de peptide ", formant une chaîne : un " polypeptide ". Une séquence protéique est une collection ordonnée de lettres choisis dans l'alphabet = {A, C, D, E, F, G, H, I, K, L, M, N, P, Q,

R, S, T, V, W, Y}. Où chacune des lettres correspond à un acide aminé. Exemple d'acide aminé : « Lysine » codé par la lettre « K ». Exemple d'une protéine : l'insuline :[17]

"F V N Q H L C G S H L V E A L Y L V C G E R G F F Y T P K A"

2.3.4 Les mutations chromosomiques

Les mutations chromosomiques désignent des processus qui provoquent un remodelage du génome à grande échelle par des modifications de la structure des chromosomes ou par des changements du nombre de copies des chromosomes dans une cellule Les mutations chromosomiques entraînent habituellement des anomalies dans le fonctionnement de la cellule ou de l'organisme qu'elles concernent. Il y a essentiellement deux raisons à cela :

- Les mutations chromosomiques peuvent aboutir à un nombre ou une position anormale(e) des gènes.
- Si la mutation implique la rupture d'un chromosome, ce qui est souvent le cas, cette rupture peut se produire à l'intérieur même d'un gène, perturbant ainsi sa fonction.[18]

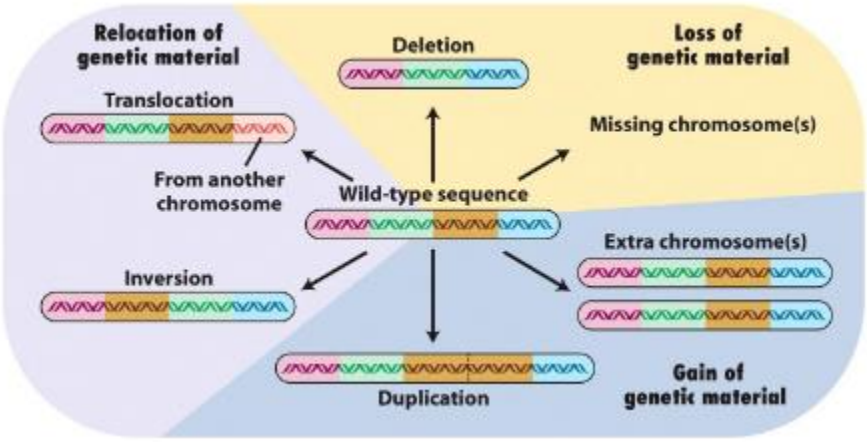

FIGURE 2.1 Les mutations chromosomiques.[18]

2.3.5 Homologie et similitude de gènes

Le paradigme central de la bio-informatique est : «la déduction par homologie ». Terminologie : Identité : proportion des paires de bases (résidus) identiques entre deux séquences exprimée généralement en pourcentage. Similitude : mesure de la ressemblance entre deux séquences. Le degré de similitude est quantifié par un pourcentage de substitutions conservatives des séquences. Homologie : deux séquences sont homologues si elles ont un ancêtre commun. Il n'y a pas de degré d'homologie. On ne dit pas : très homologues, faiblement homologues. Deux gènes sont homologues ou ils ne le sont pas. Toutes les opérations modification de gènes citées ou non dans le paragraphe précédent, permettent :[17]

- **Spéciation :** c'est la séparation d'une espèce en deux, chaque population évolue et forme une nouvelle espèce. Cette modification est le fruit d'une insertion, délétion ou mutation au niveau d'un gène.
- Les nouvelles espèces héritent des mêmes gènes, mais modifiés
- **Divergence :** leurs gènes accumulent des mutations et génèrent d'autres espèces (figure 1.16.B). Gènes Ortho logues et Para logues : Deux gènes sont homologues s'ils sont issus d'un même ancêtre. On distingue les gènes ortho logues et gènes para logues :
    - **Ortho logues :** gènes homologues et organismes différents ou espèces différentes
    - **Para logues :** gènes homologues et issus d'organismes identiques [17]

2.4 Banques de données biologiques

- Ensemble de données relatives à un domaine, organisées par traitement informatique, accessibles en ligne et à distance.
- Souvent, les données sont stockées sous la forme de fichiers texte formatés (respectant une disposition particulière).
- Besoin de développer des logiciels spécifiques pour interroger les données contenues dans ces banques.[17]

2.4.1 Banques de séquences nucléiques

Nous citons les banques les plus populaires malgré que l'accès soit toujours contrôlé via des mots de passe :

• **EMBL :** banque européenne créée en 1980 et financée par l'EMBO (Européen Moléculaire Biologie Organisation), elle est aujourd'hui diffusée par l'EBI (Européen Bio-informatiques Institute, Cambridge, UK)

• **GenBank :** créée en 1982 par la société Intelligentes et diffusée maintenant par le NCBI (National Center for Biotechnologie Information, Los Alamos, US). Elle est soutenue par le NIH (National Institute of Health). Elle possède plus de 50 millions séquences stockées

• **DDBJ :** (DNA Data Bank) créé en 1986 et diffusée par le NIG (National Institute of Génétiques, Japon) [19]. La collaboration entre les deux premières banques a commencé relativement tôt. Elle s'est étendue en 1987 avec la participation de la DDBJ. Ils ont adopté un système de conventions communes : "The DDBJ/EMBL/GenBank Feature Table Définition" en 1990 qui a défini un forma tunique pour la description des caractéristiques biologiques qui accompagnent les séquences dans les banques de données nucléiques [19]

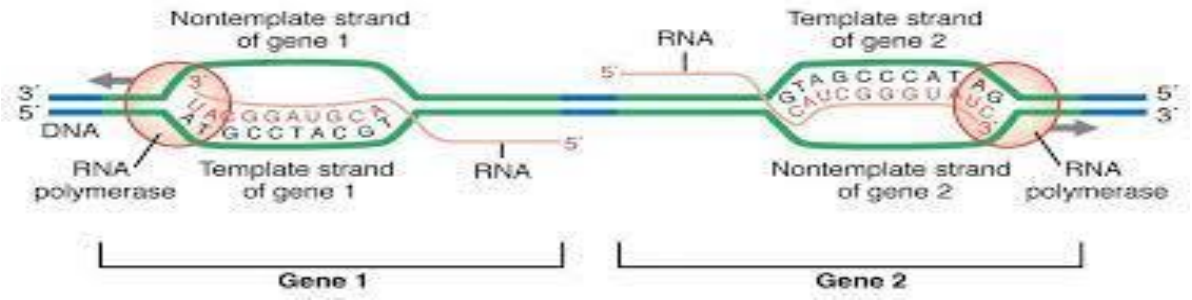

***FIGURE 0.2** Banques de séquences nucléiques*

2.4.2 Banques de séquences protéiques

PDB : La banque de données sur les protéines du" Research Collaboratory for Structural Bio-informatiques", plus communément appelée Protéine Data BankouPDB est une collection mondiale de données sur la structure tridimensionnelle (ou structure 3D) de macromolécules biologiques : protéines, essentiellement, et acides nucléiques. Ces structures sont essentiellement déterminées par cristallographie aux rayons X ou par spectroscopie RMN. Ces données

expérimentales sont déposées dans la PDB par des biologistes et des biochimistes du monde entier et appartiennent au domaine public. Leur consultation est gratuite et peut se faire directement depuis les sites web de la banque :

- Europe : PDB e ;
- Japon : PDB j ;
- États-Unis : RCSB PDB [19].

• La PDB est la principale source de données de biologie structurale et permet en particulier d'accéder à des structures 3Dde protéines d'intérêt pharmaceutique. PIR-NBRF : créée en 1984 par la NBRF (National Biomédical Research Fondation). Elle est maintenant un ensemble de données issues du MIPS (Martinsried Institute for Protéine Séquences, Munich, All emagne) et de la banque japonaise JIPID (Japan International Protéine Information Data base). SwissProt : créée en 1986 à l'Université de Genève et maintenue depuis 1987 dans le cadre d'une collaboration, entre cette université (via ExPASy, Expert Protéine Analysis System) et l'EBI. Celle-ci regroupe aussi des séquences annotées de la banque PIRNBRF ainsi que des séquences codantes, traduites de l'EMBL [19]

2.5 Problèmes issus de la bio-informatique

2.5.1 Analyse de séquences

Les données primaires des projets séquençage sont des séquences d'ADN. Celles-ci sont devenues vraiment exploitables à travers leur annotation. Plusieurs étapes d'analyse avec des outils de la bio-informatique sont nécessaires pour partir d'une séquence d'ADN crue et atteindre des séquences annotées d'une protéine :

- Établir la séquence correcte des fragments contigus d'ADN pour obtenir une séquence continue.
- Trouver les emplacements de déclenchement de transcription et la traduction, trouver des sites de promoteurs, et des ORF s (Open Reading Frame = cadre ouvert de lecture).
- Trouver emplacements d'épissage, introns, exons.

- Traduire la séquence d'ADN en une séquence de protéine.
- Comparer la séquence d'ADN à des séquences connues homologues de protéine afin de vérifier les exons... etc.
- Déterminer la structure (surtout la structure tertiaire 3D) puis la fonction de la protéine par comparaison à d'autres séquences semblables.
- Déterminer une origine et/ou une histoire évolutive commune (phylogénie).[17]

2.5.2 Recherche d'un motif dans une séquence

Un motif (ou Pattern) au sens bio-informatique du terme, représente une expression qui permet de caractériser un ensemble de séquences d'ADN, d'ARN ou de protéines. Le motif peut concerner les structures primaires, secondaires et tertiaires. Le motif trouve notamment son intérêt dans la caractérisation des fonctions des protéines : si on était capable d'exhiber un motif pour chaque fonction alors on serait en mesure de prédire automatiquement la fonction associée à une protéine. On distingue deux étapes dans la recherche de motif : [19]

• la découverte qui, étant donné un ensemble de séquences, tente d'exhiber un motif commun à ces séquences. Il s'agit d'un problème complexe car on ne sait pas ce qui doit être trouvé. Dans le cas de séquences similaires, on peut utiliser un alignement multiple des séquences afin de trouver un motif simple.

• la recherche à proprement parler, qui concerne la détection d'un motif donné sur un ensemble de séquences. Ce problème est bien plus simple que le premier. Les deux problèmes rencontrés dans la recherche de motifs concernent la définition du motif. Un motif généralement défini à partir d'un ensemble référence de séquences qui possèdent la même fonction

• si le motif n'est pas assez fin, on risque de le découvrir sur des séquences qui n'ont pas la fonction liée au groupe de séquences référence, ces séquences seront appelées faux positifs

• par contre, s'il est trop fin, certaines séquences qui possèdent la fonction liée au motif ne seront pas découvertes, on les qualifiera de vrais négatifs.

2.5.3 Phylogénie

La phylogénie ou reconstruction phylogénétique peut être définie comme la reconstruction de l'histoire évolutionnaire d'un ensemble d'espèces. L'évolution entre les espèces est représentée sous forme d'un arbre (phylogénétique) dont les branches indiquent le degré de proximité entre les espèces. Il existe deux façons de caractériser les espèces :

- Soit en se basant sur leurs phénotypes, c'est à dire l'ensemble des caractéristiques qu'elles expriment (suivant leur apparence).
- Soit en se basant sur leurs génotypes, c'est à dire l'ensemble des caractéristiques comprissent dans leurs génomes et qu'elles peuvent éventuellement exprimer.

Le premier genre d'approche basé sur les phénotypes est utilisée lorsqu'on ne dispose d'aucune information sur les génomes des espèces. Depuis le lancement des différents programmes de séquençage des génomes on a de plus en plus tendance à utiliser le second type d'approche. Ici nous ne nous focaliserons que sur le second type de caractérisation. Nous utiliserons des séquences nucléotidiques (ou d'acides aminés) pour distinguer les espèces entre elles Il existe 3 types d'approches pour s'attaquer au problème de reconstruction phylogénétique

Les approches basées sur les distances utilisent une mesure de distance ou de similarité afin de regrouper des séquences proches, en anglais on parle de clustering. Ces approches sont très efficaces car elles sont basées sur un algorithme de complexité polynomiale ce qui les rend particulièrement adaptées pour des études à grande échelle

Les approches axées sur les caractères cherchent le meilleur arbre possible dans une topologie d'arbre étant donné un critère d'optimalité. Le critère le plus largement utilisé est le critère de maximum de parcimonie (Maximum Parsimony Criterion) pour lequel on considère que le meilleur arbre est celui qui requiert le minimum de changements. Le problème de maximum de parcimonie est un problème NP-dur dans le cas général, il est donc nécessaire de faire appel à des techniques heuristiques et d'optimisation afin de le traiter efficacement dans un temps raisonnable [19]

2.5.4 Comparaison de séquences

La recherche de similitude entre séquences est une tâche importante dans l'analyse des séquences. Elle est la base de plusieurs autres problèmes bio-informatiques comme la

construction des arbres phylogénétique, la recherche des motifs, l'assemblage de fragments d'ADN, etc. Mais cette tâche n'est pas facile car elle nécessite l'élaboration de procédures de calcul efficaces et le modèle utilisé doit être biologiquement acceptable. Parmi les problèmes traités on peut citer la recherche des segments identiques entre plusieurs séquences, la recherche des régions significatives comme les introns et les prometteurs, la recherche de la super séquence minimale qui englobe un ensemble de séquences, l'alignement multiple de séquences, etc. La plupart des problèmes de comparaison de séquences ont été démontré NP-difficile. Même une solution dont la complexité est polynomiale est difficile à résoudre en bio-informatique vu le grand nombre et la taille colossale des séquences à traiter.[20]

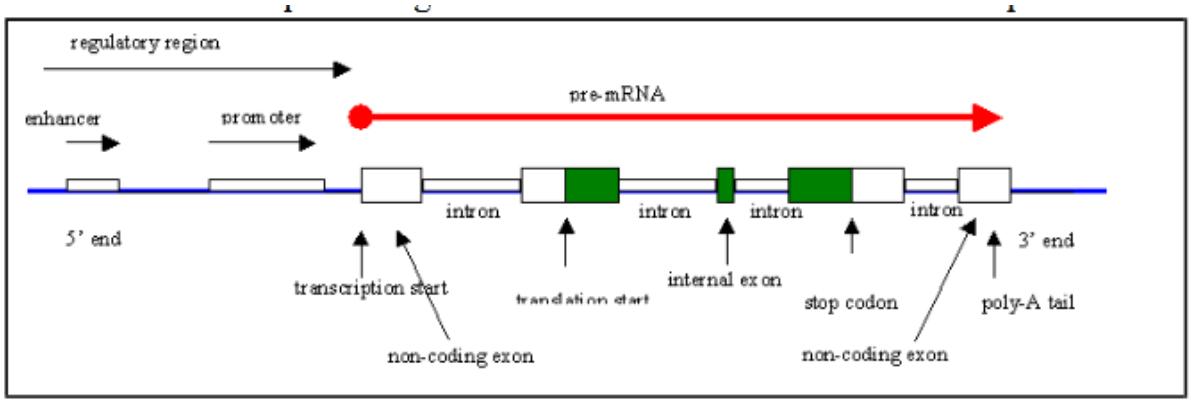

FIGURE 0.3 *Structure d'un gène eucaryote [20]*

2.5.2. Alignement de séquences

L'alignement de séquences est une problématique importante de la bio-informatique. En effet aligner des séquences constitue un problème à part entière, mais il est également utilisé comme point de départ pour d'autres problèmes de bio-informatique. Le problème de l'alignement de séquence sera détaillé dans les deux chapitres suivante de ce manuscrit.[21]

3. Conclusion

Dans ce chapitre, nous avons passé en revue quelques notions de base de la bio-informatique. Ainsi que les principaux problèmes de ce domaine.

Dans le chapitre suivant, nous détaillons le problème étudié dans ce travail, à savoir, le problème d'alignement multiple des séquences.

.

Chaptire3 : Alignement multiple des séquences

3.1 Introduction

L'alignement multiple des séquences d'ADN ou de protéines est une des techniques les plus utilisées dans l'analyse de séquence. Il est considéré parmi les problèmes les plus difficiles en bio-informatique. L'alignement multiple de séquences (Multiple Séquence Alignement : MSA) est une tâche cruciale et très importante en biologie moléculaire. MSA offre aux biologistes un moyen pour analyser des séquences d'ADN ou de protéines et de déterminer par la suite leur degré d'homologie ou de divergence [17]

L'alignement multiple de séquences permet de mettre en évidence les similarités entre plusieurs séquences. Il est donc possible de comparer simultanément la proximité de toutes ces séquences. Les informations apportées par ces comparaisons permettent d'obtenir des renseignements importants sur les séquences comme les distances d'une séquence par rapport aux autres ou encore la mise en évidence de zones identiques entre plusieurs ou toutes les séquences [19]

3.2. Alignement par paires de séquences

L'alignement par paire de séquences de protéines est un outil fondamental de la bio-informatique. Il a pour but principal de faire ressortir les séquences apparentées, en mettant en évidence les régions communes. L'alignement par paire est principalement utilisé pour la comparaison d'une séquence avec un ensemble de séquences. Les algorithmes$Fasta$ et $Blast$ permettent de comparer une séquence à un ensemble de séquences contenues dans une base de données. Une séquence peut être définie comme une suite finie et ordonnée de lettres prises dans un alphabetΣ. Pour les protéines, cet alphabet est constitué de 20 $lettres$, appelées acides aminés ou résidus.[22]

3.2.1. Définition

Soient $S1$= ($x11$, $x12$..., $x1|S1|$) et $S2$= ($x21$, $x22$..., $x2|S2|$) deux séquences définies sur un alphabet Σ. Un alignement A de S1 et S2 est une matrice de caractères de Σ∪ {−} définie par [22]:

A= [$a11$, $a12$..., $a1$

$qa21$, 22...,$2q$]

Et vérifiant les propriétés :

- Max ($|S1|, |S2|$) $\leq q \leq |S_1|+|S_2|$,
- $a_{ui}=x_{u,v}$ ou $-$, $\forall u \in \{1,2\}, \forall v \in [1..|S_u|]$
- $\nexists i$ tel que $a_{1i}= a_{2i}= -$

3.2.2. La distance entre deux séquences

3.2.2.1. Similarité et homologie

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

- Suppression d'un ou plusieurs nucléotides,
- Insertion d'un ou plusieurs nucléotides,
- Mutation d'un nucléotide en un autre. Au sens de la théorie de l'évolution, deux séquences peuvent donc être plus ou moins proches, selon le nombre de modifications ayant eu lieu. [23]

On dit qu'il y a homologie entre deux séquences lorsque celles-ci possèdent une parenté du point de vue de l'évolution. On dit qu'il y a similarité de séquences, lorsqu'il y a de nombreuses identités entre les séquences. Pour les protéines, cela se caractérise par des paires de résidus composées de deux acides aminés appartenant à la même famille physico-chimique.[23]

3.2.2.2. La distance de Hamming

Définition 1 : Soient S et T deux séquences sur un alphabet Σ. On dit que deux lettres si et ti de S et T se correspondent si, et seulement si $S_i=T_i$. On utilise également le terme anglais match pour indiquer la correspondance de deux lettres.[23]

Définition 2 : Soient S et T deux séquences de même longueur n sur un alphabet Σ. La distance de Hamming entre S et T, notée dH (S, T), représente le nombre de caractères S[i] et T[i] qui ne se correspondent pas. La distance de Hamming est définie par :[23]

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

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

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

Exemple : Soient S et T les deux séquences suivantes :

S : ACACACAT

T : CACACA

La distance de Hamming entre S [1...7] et T est donc d_H(S [1..7], T) = 7, alors que la distance entre S [2..8] et T est d_H (S[2..8], T) = 0. [23]

3.2.3. Les Matrices de Substitution

Le choix d'une matrice de substitution gouverne le système des scores et par conséquent influe sur les résultats obtenus. Il existe deux types de matrices de substitution à utiliser selon la nature des séquences nucléiques ou protéiques

3.2.3.1. Matrices de Scores pour l'ADN

La matrice Identité : Cette matrice consiste en l'attribution d'un score 1 en cas d'identité sinon un zéro.[20]

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

FIGURE 0.1 *Matrice de Score Pour l'ADN.[20]*

La matrice de Transition/Transvasions : Dans cette matrice en prend en considération l'effet des actions des transitions (A à G, G à A, C à T, et T à C) et Transvasions (les autres passages entre

–	A	C	G	T
A	3	0	1	0
C	0	3	0	1
G	1	0	3	0
T	0	1	0	3

FIGURE 0.2 *Matrice de transition.[20]*

nucléotides) Identité=3Transition= 1, Transvasions= 0.[20]

• La matrice BLAST : La matrice identité Blast. C'est une matrice de même principe que la matrice Identité sauf que les valeurs attribuées en cas d'identité et substitution sont différentes de 1 et 0. On Remarque que la substitution ici est fortement pénalisée.[15]

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

FIGURE 0.3 *La matrice BLAST.[20]*

3.2.3.2. Matrices de score pour les protéines

• **Matrices PAM**

Dans une matrice PAM, le score donné à une paire d'acides aminés est la mesure de probabilité du changement d'un acide aminé à un autre dans une famille de protéines. D'abord, un concept des unités PAM a été introduit pour mesurer la quantité de changement évolutionnaire dans une séquence de protéines. Deux séquences S et T sont définies pour être un PAM unité divergé si une série de points de mutations admises (sans l'insertion ou la suppression) peut convertir S à T avec une moyenne d'une substitution d'un acide aminé par cent acides aminés [15]. Afin de créer la matrice PAM1, des séquences convergentes dans une famille de protéines ont été alignées. Pour chaque pair d'acides aminés, la fréquence des substitutions entre ces deux acides aminés dans un alignement a été déterminée. Ces probabilités ont été placées dans une matrice représentant tous les changements possibles d'acides aminés. Ensuite, la matrice a été normalisée en des valeurs qui représentent la probabilité qu'un d'acide aminé parmi cent subit le changement (unités de PAM). Pour les pairs d'acides aminés (a, b), soit M (a, b) la fréquence observée de substitution, et P la fréquence prévue, donc :

$$\text{Score (a, b)} = 10 \log10 \text{ (M (a,) de b/P)}$$

Les scores sont arrondis jusqu'au prochain nombre entier comme indiqué dans la table 1.3. Un score positif indique que les substitutions entre les acides aminés a et b sont plus probables. Un score nul indique que les substitutions entre a et b se produisent à un taux de base aléatoire.

Un score négatif indique que les substitutions entre a et b sont moins probables. Une série de matrices de PAM, telles que PAM160 ou PAM250, a été construite à partir de la matrice PAM1. PAM250 est la matrice de la famille PAM la plus recommandée pour effectuer des alignements.

• **Matrice BLOSUM**

Dans les matrices de BLOSUM, des scores pour chaque pair d'acides aminés sont déterminés par des observations des fréquences de substitutions dans les blocs d'alignements locaux des protéines reliés dans la base de données BLOCS. Dans BLOCS il existe 3.000 blocs de séquences fortement conservées représentants des centaines de groupes de protéines. De même que la construction des matrices de PAM, les logs de probabilité de la fréquence de substitution pour chaque pair d'acides aminés sont calculés pour établir une matrice de BLOSUM. Il y a une série de matrices BLOSUM, chaque BLOSUMn dénoté pour un certain n, où le nombre n'indique le degré de similarité des séquences desquels la matrice de BLOSUM a été dérivée [15]. Par exemple, la matrice BLOSUM30 est créé à partir des alignements de séquences partageants pas plus de 30% d'identité. On pense que la matrice BLOSUM62 est une bonne matrice globale tandis que BLOSUM45 est recommandée pour des séquences plus divergentes et BLOSUM100 est suggéré pour des séquences fortement liées [15]. La matrice BLOSUM62 a été trouvée plus semblable à la matrice PAM250 mais elle est plus efficace pour trouver les membres des familles de protéines.

3.2.4 Opérations d'édition

L'alignement de deux séquences consiste à mettre en regard tous les caractères composant ces deux séquences. Les opérations d'éditions définissent les différentes modifications nécessaires permettant d'expliquer l'évolution de la séquence S jusqu''a la séquence T. Ainsi, pour chaque paire de résidus, quatre cas sont possibles, correspondant chacun `a une opération d'édition :

- L'appariement ou match qui correspond à deux caractères qui se correspondent :(a, a),

S: AGTGAGG S′: AG--TGAGG

T: AGGTTGCG T′: AGGTTG-CG

Exemple d'alignement.

La substitution ou mis match qui correspond à deux caractères qui ne se correspondent pas : (a, b) avec a≠b

– l'ajout d'une brèche dans S ou insertion (−, b),

– l'ajout d'une brèche dans T ou délétion (a, −). L'évolution de la séquence S jusqu''a la séquence T s'obtient en réalisant des délétions a la place des insertions, et réciproquement.[23]

3.2.5 Evaluation des brèches

L'évaluation des brèches dans un alignement est très importante. Selon les valeurs attribuées, le nombre de brèches peut varier. Si elles ne sont pas assez pénalisantes, la fonction de score risque de favoriser les alignements qui en contiennent beaucoup. En particulier, en morcellent les séquences il est possible d'augmenter le nombre de correspondances. Inversement, si les brèches sont trop fortement évaluées, les alignements ne comportant pas assez de brèches sont favoris´es, empêchant ainsi d'avoir toutes les correspondances. Il existe principalement deux modélisations [23] pour évaluer le cout engendré par l'insertion d'une brèche. Les valeurs associées à une brèche pour ces modèles peuvent être obtenues par des fonctions. Ces fonctions prennent en paramètre la longueur de la brèche et retourne le cout de celle-ci.

3.2.5.1 Les brèches à cout constant

Ce modèle de calcul pour les brèches est le plus simple, puisqu'il attribue la même valeur `a tous les caractères '−'. Ainsi, si la valeur associée à une brèche de longueur 1 est α, le cout global associé à une brèche de longueur L quelconque est α .l

Ce modèle permet une évaluation très simple de l'alignement. En effet, comme toutes les positions d'une brèche ont la même valeur, l'évaluation repose sur le même principe que pour les paires de résidus. Il suffit pour cela de faire la somme de toutes les valeurs sur la longueur de

l'alignement. Le symbole '−' peut alors être considéré comme un acide aminé pour l'évaluation.[23]

3.2.5.2 Les brèches à cout affine

Selon les biologistes, l'évaluation `a cout constant même si elle est simple à mettre en œuvre a un défaut majeur. Elle n'est en effet pas représentative de la façon dont les choses se passent dans la réalité. D'un point de vue biologique, la création de la brèche est beaucoup plus pénalisante que son élongation. Il semble alors normal de prendre en compte cette remarque importante, et donc d'associer un cout différent suivant qu'il s'agit du début ou de l'extension de la brèche. Les dénominations généralement utilisées pour le cout d'ouverture d'une brèche sont K ou gop (gap opening penalty). Pour l'extension de la brèche on utilise fréquemment h ou gep (gap extending penalty). Le cout associé à une brèche de longueur lest alors la fonction affine gap(l) =K+h.(l−1). On trouve également cette fonction sous la forme gap(l) =K′+h . l en posant K′= K − h.

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

- La première consiste à n'évaluer que les paires de résidus, puis à y ajouter la somme des valeurs de toutes les brèches.
- La seconde méthode consiste à évaluer l'ensemble de l'alignement en faisant la somme de toutes les positions. [23] Cette méthode nécessite lorsque l'on doit évaluer une brèche de savoir si la position précédente était également une brèche ou non. En pratique, seule cette méthode est utilisée pour la construction de l'alignement, même si elle génère plus de calculs que dans le cas des brèches à coût constant. Les valeurs généralement utilisées pour des matrices de substitution standard sont de l'ordre de −10 pour K et−1 pour h. Ces valeurs peuvent bien sûr varier, mais en conservant toujours un ratio K/h voisin de 10. [23]

3.2.6 Méthodes d'Alignement par paires

Il existe deux types d'alignements de séquences : global et local. Le premier prend en considération l'ensemble des résidus de chacune des séquences. Si les longueurs des séquences sont différentes, alors la plus courtes va subir des insertions de gaps afin d'arriver à aligner les deux séquences d'une extrémité à l'autre. Cependant dans un alignement global, si uniquement des segments courts sont très similaires entre deux séquences, les autres parties des séquences risquent de diminuer le poids de ces régions. C'est pourquoi d'autres algorithmes d'alignements, dits locaux, basés sur la localisation des zones de similarité sont nés. Le but de ces alignements locaux est de trouver sans prédétermination de longueur les zones les plus similaires entre deux séquences. L'alignement local comporte donc une partie de chacune des séquences et non la totalité des séquences comme dans la plupart des alignements globaux.[17]

3.2.6.1 Alignement Global

Plusieurs méthodes ont été développées afin de réaliser un alignement global de deux séquences le plus correct que possible. Parmi ces méthodes et qui sont toujours utilisées on trouve des méthodes graphiques et autres qui utilisent la programmation dynamique

- **Algorithme Needleman & Wunsch**

Basé sur la programmation dynamique (la récursivité), cet algorithme ne calcule pas la différence entre deux séquences mais la similarité. Considérons deux séquences $A\ (1, n)$ $B\ (1, m)$ [4]. Un tableau à deux dimensions est rempli ligne après ligne (en partant de la dernière) et pour chaque ligne, colonne après colonne (en partant de la dernière) en obéissant à la règle suivante : Le score $S\ (i, j)$ est le nombre maximum de correspondance entre les deux parties de séquences $A\ (i, n)$ et $B\ (j, m)$ (en prenant tous les chemins possibles à partir de (i, j)) et en appliquant une fonction de score :[17]

• score pour une identité =1.

• score pour une substitution, une insertion ou délétion =0.

La formule de récurrence est

$$S(i,j) = \max \begin{cases} \text{Si } ai = bj+1 \quad S(i, j+1) -1 + s(ai, bj), \quad \text{sinon } S(i, j+1) + s(ai, bj) \\ \text{Si } ai+1 = bj+1 \quad S(i+1,j+1) -1 + s(ai, bj), \quad \text{sinon } S(i+1,j+1) + s(ai, bj) \\ \text{Si } ai+1 = bj \quad S(i+1,j) -1 + s(ai, bj), \quad \text{sinon } S(i+1, j) + s(ai, bj) \\ \text{Avec évidemment} \quad S(n+1, j) = S(i, m+1) = 0 \end{cases}$$

3.2.3.3. Alignement Local

Ce type d'alignement est favorable aux séquences divergentes car un alignement global serait non significatif. Pour l'alignement Local, Smith et Waterman une méthode exacte qui permet d'aligner deux séquences en essayant d'aligner des segments communs ou motifs.[17]

- **L'algorithme Smith &Waterman**

L'algorithme de Smith et Waterman est décrit pour l'alignement local de deux séquences. Il identifie les sous séquences maximales de deux séquences par programmation dynamique [17]. La différence essentielle de cet algorithme avec l'algorithme de Needleman et Wunsch est que n'importe quelle case de la matrice de comparaison (initiale) peut être considérée comme point de départ pour le calcul des scores sommes et que tout score somme qui devient inférieur à zéro stoppe la progression du calcul des scores sommes et il sera réinitialisé par la valeur 0. La case concernée peut être considérée comme nouveau point de départ. Cela implique que le système de score choisi possède des scores négatifs pour les mauvaises associations qui peuvent exister entre les éléments des séquences. L'équation utilisée pour le calcul de chaque score somme pendant la transformation [17] de la matrice initiale prend alors l'expression suivante :

$$S(i,j) = \max \begin{cases} Sc(i,j) + S(i+1, j+1) \\ Sc(i,j) + \max S(x, j+1) - P \\ Sc(i,j) + \max S(i+1, y) - P \\ 0 \qquad \text{avec } i+2 < x < m \text{ et } j+2 < y < n \end{cases}$$

3.3. Alignement multiple de séquences

3.3.1 Évaluation

Ayant obtenu un alignement multiple donné, celui-ci va être évalué paire par paire de séquences. L'idée est de comparer chaque paire de résidus de l'alignement de deux de séquences avec les paires de résidus contenus dans la bibliothèque.[17]

3.3.2 Les approches de résolution

Dans la littérature, on rencontre trois catégories essentielles ou approches suivies pour construire un MSA. Néanmoins, ces approches sont parfois fusionnées, concaténées ou/et associées pour construire une seule méthode. On distingue l'approche Exacte qui tente de donner plus de longévité à la programmation dynamique dans ce domaine et de déterminer un alignement optimal proprement dit comme elle le fait pour aligner deux séquences. De l'autre côté, on rencontre des heuristiques qui à leur tour se bifurquent en deux approches : Progressive et Itérative [17]. Les méthodes qui suivent l'approche progressive, sont reconnues d'être très rapides et donnent des résultats assez satisfaisants mais leur inconvénient est le fait de s'arrêter sur les minima locaux et si une erreur est commise au début de l'alignement, elle va se propager sur l'alignement final. L'approche itérative est une manière très simple, rapide et efficace permettant d'améliorer des méthodes d'alignement multiples. L'itération peut être employée pour améliorer le résultat d'un logiciel existant avec n'importe quelle fonction objective. Elle peut également être incorporée à une stratégie progressive d'alignement pour établir des alignements à partir de zéro pour produire encore de meilleurs résultats. [17]

3.3.2.1 L'Approche Exacte

L'approche exacte n'est autre qu'une généralisation des méthodes de programmation dynamique de La méthode de programmation dynamique utilisée pour aligner deux séquences, a été appliquée à l'alignement de plusieurs séquences (N dimensions) tels que MSA et DCA. Ce type de méthodes représente de gros problèmes : Le temps de calcul et l'espace mémoire.

• Dans la pratique, un alignement devient délicat pour un nombre de séquence N>3, et même impossible pour N=10.[17]

• Pour Nséquences de longueur L, l'alignement optimal (au sens mathématique) nécessite :

- Un temps de calcul proportionnel à 2n Ln
- Un espace mémoire proportionnel à Ln

Exemple :

Pour 10 séquences de 100 résidus, et 10-9 secondes de temps de calcul par colonne, nécessite alors : Temps total = $210\times10010\times10-9\approx1014s(>910années)$ Espace mémoire : 10 11 6 GB.

Le problème de l'alignement multiple exacte a été démontré être un problème NP-complet. D'où le recours aux méthodes approchées ou heuristiques [17]

3.3.2.2 L'Approche Itérative

L'approche itérative a été employée plusieurs fois comme méthode d'optimisation pour produire des alignements multiples. Parfois elle est utilisée seule ou en combinaison avec d'autres méthodes. L'itération a un grand avantage parce qu'elle est souvent très simple soit en termes de code Des algorithmes soit en termes de complexité temporelle et spatiale.[17]

3.3.2.3 L'Approche Progressive

L'alignement progressif est l'heuristique la plus répandue pour aligner un grand nombre de séquences. L'alignement multiple est construit progressivement en alignant des paires de séquences suivies des paires d'alignements/profils. Un arbre guide détermine l'ordre dans lequel les séquences vont être alignées, les plus proches d'abord. Cette technique est employée dans différents packages d'alignement multiple tels que, ClustalW, et T-Coffee ...etc. [17]

Un alignement multiple progressif suit les étapes suivantes :

- Alignement deux à deux de toutes les séquences
- Construction d'une matrice de distances entre toutes les séquences.
- Détermination de l'ordre selon lequel les séquences seront alignées en utilisant la notion de clustering :
  - Alignement de deux séquences.

- Alignement d'une séquence et d'un profil.

Problèmes majeurs des alignements multiples progressifs

- Les alignements entre sous-groupes sont gelés. Si une erreur est produite au début, aucune modification ou correction ultérieure n'est possible
- Les erreurs dans les alignements des sous-groupes initiaux se propagent dans tous l'alignement.[17]

3.3.3 Étude Comparative des Méthodes

Le nombre des méthodes d'alignement multiple de séquences qui ont été développées reflète l'importance des alignements multiples dans l'analyse de séquence de jour en jour et la variété des buts pour lesquels elles sont nécessaires. Ceci soulève la question quelle méthode faut-il employer et quels critères devraient être employés en comparant des méthodes. ClustalW [17] est l'une des méthodes les plus répandues et de ceci reflète son interface graphique facile à utiliser. D'autres méthodes sont plus rapides et/ou plus précises MAFFT [17] et MUSCLE Quelques packages sont relativement lents mais semblent donner des alignements d'un niveau d'exactitude assez élevé ou acceptent en entrée des données hétérogènes ; T-Coffee [17] et ProbCons. Autres méthodes sont spécialisées pour aligner des génomes entiers MLAGAN tandis que d'autres sont spécialisées pour des alignements multiples locaux DIALIGN et Align_M. Toutes les méthodes présentées dans la section précédentes ont certainement des avantages et des inconvénients. Leurs défaillances sont dues généralement soit au choix de l'approche elle-même, soit au choix de la fonction objectif à optimiser, ou bien tout simplement à la complexité des données biologiques traitées. Par contre, toutes ces méthodes exposent toujours une nouvelle idée et un nouvel état d'esprit dans la prise en charge de l'analyse biologique. Toutes les facettes de cette analyse ont été prises en charge par l'une ou l'autre méthode. [17]

3.3.4 Evaluation des algorithmes d'alignement

Le critère d'évaluation utilisé est souvent une fonction de score. Il est donc nécessaire de pouvoir évaluer la qualité de chaque solution. Quelques fonctions ont été définies afin de permettre une évaluation des alignements, mais elles peuvent également être utilisées par des algorithmes lors de la construction de ces alignements.[23]

3.4 Conclusion

Dans ce chapitre, nous avons exposé le problème d'alignement multiple des séquences MSA qui est l'objet de notre étude.

Nous présentons dans le chapitre suivant les méthodes de résolution proposées pour résoudre ce problème.

Chapitre4 : Conception

4.1 Introduction

Dans ce chapitre, nous discutons notre méthode proposée pour résoudre le problème d'alignement multiple de séquences.

La méthode proposée est un algorithme génétique amélioré. Cette amélioration est effectuée en appliquant l'algorithme de recuit simulé sur la meilleure solution (le meilleur alignement) trouvée par l'algorithme génétique. L'objectif dans chaque itération est de minimiser le nombre de dissimilitudes entre les séquences.

4.5.Codage

L'étape clé dans l'implémentation des alignes est bien le codage des solutions. Celui-ci consiste à passer de la représentation réelle des solutions vers une représentation codée. Le but du codage des solutions est de rendre les solutions plus maniables par l'algorithme de recherche.

Dans notre problème nous avons représenté les solutions par des matrices de M*N tel que :

N : nombre de séquence

M : taille d'un alignement

Chaque séquence représente une protéine (suite des acides aminés) comme le montre le schéma suivant :

"WGKVNVDEVGGEAL

"WDKVNEEEVGGEAL

"WGKVGAHAGEYGAEAL

"WSKVGGHAGEYGAEAL"

Est une population

4.6.Evaluation des solutions

C'est la fonction la plus répandue, elle consiste à sommer les scores de paires de séquences alignée dans un alignement multiple Ai. Soit Ai un alignement de K séquences S_1..., S_k

$$(SP(A_i)) = \sum_{j=i+1}^{k-1} \mathrm{Sc}(S_i S_j)$$

Sc (S_iS_j) : le score de l'alignement de la paire des séquences Si et Sj (calculé par une mesure de distance ou de similitude).

Exemple : soit l'alignement A suivant :

S1: a c - c d b -

S2: - c - a d b d

S3: a - b c d a d

Si on considère que la fonction des distances est :

d(x,x)=0, d(x,y)=1 pour x≠y

Le SP de alignment A = Sc (s1, s2) + Sc (s1, s3) +Sc (s2, s3) = 3+4+5 =12.

4.7.Pseudo code de l'AG amélioré pour résoudre le problème d'alignement

Algorithm 1 Alignement multiple en utilisant l'algorithme génétique

Input: *Originale_Séquences*[*N*] : Les N séquences originales non alignées -sans gaps- a alignées
Input: *Taille_Population* : taille de population
Input: *Nb_itération* : nombre des itérations
Input: *Croisement_probabilité* : Le taux de croisement
Input: *Mutation_probabilité* : Le taux de mutation
Output: *Alignement_Optimale* la meilleure solution trouvée par l'algorithme génétique

```
1: Population ← {∅}
2: Fitness_List[N] ← {}
3: ▷ liste des meilleur Solutions(scores) pour chaque génération
4: Meilleur_Solution[Nb_itérations] ← {}
5: ▷ Initialisation de la première génération
6: Population ← Init_Génération(Originale_Séquences,Taille_Population)
7: for i from 1 to Nb_itération do
8:     for j from 1 to N do
9:         ▷ Évaluation de la population courante
10:        Fitness_List[j] ← Évaluation(Population[j])
11:    end for
12:    ▷ Ajouter la meilleur solution de la génération courante
13:    Ajouter(Meilleur_Solution,(Min(Fitness_List),Clé=Score))
14:    Individus_sélectionnés ← roulette_sélect(Population)
15:    if Mutation_probabilité > random1() then
16:        Mutation(Individus_sélectionnés)
17:    end if
18:    if Croisement_probabilité > random2() then
19:        Croisement(Individus_sélectionnés)
20:    end if
21:    if random2() > Mutation_probabilité and random1() > Croisement_probabilité then
22:        Passer à la génération suivante
23:    end if
24: end for
25: ▷ Retourner la meilleure solution qui contient le minimum score dans la liste des Meilleures solutions
26: Return Min(Meilleur_Solution,Clé=Score)
```

Algorithme2 *Alignement multiple en utilisant l'algorithme génétique.*

4.7.1. Générâtes de la population initiale :

Algorithm 1 Initialisation de la population génération 0

Input: *Originale_Séquences*[*N*][] : N séquences originales sans alignements ou la taille de chaque séquence est différente
Input: *Taille_Population* : La taille de la population
Output: *Gen_0* : La population de génération 0 qui contient des individus et tous les séquences sans alignées

```
▷ Trouver la taille maximale pour alignées tous les séquences
Max_length ← max(Originale_Séquences,Clé=Taille)+2
Gen_0← {∅}
for i from 1 to Taille_Population do
    Individu[N] ← {}
    for j from 1 to N do
        ▷ Trouves les nombres des gaps nécessaire pour alignée la séquence j
        Nb_Gaps ← Max_length -Taille(Originale_Séquences[j])
        ▷ Ajouter les NB_Gaps dans des positions aléatoires dans la séquence
        Individu[j] ← Add_Random_Gaps(Originale_Séquences[j],Nb_Gaps)
    end for
    Gen_0 ← Gen_0+{Individu}
end for
```

Algorithme3 *Initialisation de la population de génération 0.*

4.7.1.1.Evaluation

Algorithm 1 Évaluation d'un alignement

Input: *Individu*[*N*][*M*] : Un individu contient N séquence de taille M
Input: *Gap_Open_Penalty* : la valeur de punissions d'ouverture de gap
Input: *Gap_Extension_Penalty* : la valeur de punissions d'extension de gap
Output: *Score_Individu* : la valeur d'évaluation d'un individu

```
Sum ← 0
for i from 1 to N-1 do
    for j from i+1 to N do
        for k from 1 to M do
            A ← Individu[i][k]
            B ← Individu[i][k]
            if A est Acide Aminé and B est Acide Aminé then
                if A!=B then
                    Sum ← Sum+1
                end if
                Gap_Open ← FALSE
            else
                if Gap_Open =FALSE then
                    Sum ← Sum+ Gap_Open_Penalty
                    Gap_Open ← TRUE
                else
                    Sum ← Sum+ Gap_Extension_Penalty
                end if
            end if
        end for
    end for
end for
Return Sum
```

Algorithme4 *Évaluation d'un alignement.*

Cette évaluation va être utilisée pour la sélection des individus pour les opérations de croisement et de mutation.

4.7.2. La Sélection

Il existe plusieurs techniques de sélection. Voici les principales utilisées : Sélection par rang, Probabilité de sélection proportionnelle à l'adaptation, Sélection par tournoi, Sélection uniforme

Parmi ces méthodes, nous avons choisi la méthode de sélection du tournoi en raison de sa facilité, de son utilisation fréquente et de son efficacité (la méthode est expliquée dans le premier chapitre).

4.7.3. Le Croisement

C'est une opération classique des algorithmes génétiques. Elle consiste en la combinaison du matériel génétique de deux individus de la population et en produire deux nouveaux individus (Enfants ou Descendants) héritant des caractéristiques de leurs parents. Dans notre cas, on prend deux alignements de la population secondaire, on coupe ces deux alignements chacun en deux parties. La position du croisement sur les deux alignements est tirée au hasard. Les quatre parties obtenues vont servir pour former deux nouveaux alignements. Les nouveaux alignements peuvent être insérés dans la population de la génération courante

Exemple

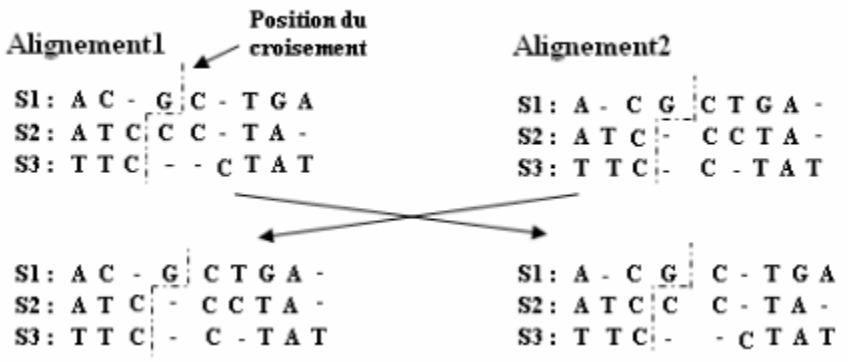

FIGURE 0.1 *Le croisement entre deux alignements*

4.7.4. La Mutation

Ayant un alignement la mutation consiste à permuter un gap et un résidu dans une séquence ou un ensemble de gaps consécutifs avec un autre de résidus. La taille d'un ensemble ici est une condition nécessaire. Les éléments à permuter doivent être voisins dans la même séquence. L'opération de mutation consiste donc à apporter une certaine diversification dans la population, elle est perçue comme une technique d'exploration de l'espace de recherche. Cette opération s'effectue sur une ou plusieurs séquences d'un alignement, elle introduit plus de diversité dans

les alignements nouvellement produits. Comme les alignements de notre population initiale sont générés par des méthodes différentes, ils peuvent ne pas être tous alignés dans le même ordre de séquences, par conséquent il arrive que l'on ne puisse pas effectuer l'opération du croisement. Dans une telle situation et pour atteindre la taille maximale de la population, la mutation devient le seul outil capable de produire de nouveaux alignements à partir de ceux déjà crées en y introduisant une modification.

Exemple

FIGURE 0.2 *Mutation d'une position*

4.8.Recuit simulé

On applique l'algorithme de recuit simule après l'exécution de l'algorithme génétique pour améliorer la qualité de la meilleure solution trouvée selon l'algorithme suivant :

Algorithm 1 Alignement multiple en utilisant l'algorithme génétique+ Recuit Simulé

Input: *Meilleur_Solution* : la meilleure solution trouvée par l'algorithme génétique.
Input: *Seuil*,*Température* : Paramètres de recuit simulé.
Output: *Meilleure_Solution_RS* :la meilleure solution trouvée par le recuit simulé.

1: $S\_Courante \leftarrow Meilleur\_Solution$
2: **do**
3: ▷ *Créer une nouvelle solution(voisinage) a partir de la solution courante*
4: $S_i \leftarrow$ Créer_Voisinage($S\_Courante$)
5: ▷ *Calculer le taux de score entre les deux solutions*
6: $Taux \leftarrow \frac{S_i - S\_Courante}{S\_Courante}$
7: **if** Score($S\_Courante$) < Score(S_i) **and** ABS($Taux$) >random() **then**
8: ▷ *Condition ou la nouvelle solution est meuilleur que la solution courante (moins de score)*
9: $S\_Courante \leftarrow S_i$
10: **end if**
11: $Température \leftarrow Température$ - 1
12: **while** $Température \geq$Seuil

Algorithme 5 Algorithme d'alignement multiple en utilisant AG+RS

4.8.1. Méthode de « Créer Voisinage »

La méthode Créer voisinage parmi de trouver une autre solution à partir de la solution courante par faire une simple modification au niveau séquences (changement d'un acide aminé).

FIGURE 0.3 *voisinage d'une solution*

4.9.Conclusion

Dans ce chapitre, nous avons présenté comment on a adapté l'algorithme génétique et le recuit simulé pour résoudre le problème d'alignement multiple des séquences.

L'approche proposée est une amélioration de l'algorithme génétique classique. Le chapitre suivant présentera nos expérimentations et nos résultats.

Chapitre 5 : Implémentation et résultats

5.4.Introduction

Après avoir expliqué et détailler la conception générale de notre travail dans le chapitre précèdent, le passage à l'étape suivante dite implémentation nous est obligatoire pour aboutir à nos objectifs. Dans ce chapitre final, nous allons présenter les différents outils de développement ainsi que les langages appris et exploités dans la réalisation de notre projet. Après, nous réaliserons un exemple illustratif sur notre application pour l'algorithme traité ultérieurement (alignement multiple). A la fin, nous exposerons et discuterons quelques résultats expérimentaux

5.5.Langage de programmation et outils de développement

Pour pouvoir réaliser notre projet nous avons fait recours à des outils et des langages adéquats qui étaient choisis soigneusement. Ces derniers sont énumérés avec une brève définition, pour chacun d'eux, dans ce qui suit :

5.5.1. Langage de programmation Python

Python est un langage de programmation, créé par Guido van Rossumont, dont la première version est apparue en 1991. Python est un langage puissant, à la fois facile à maîtriser et riche en possibilités. Dès son installation sur ordinateur par exemple, l'utilisateur dispose d'une multitude de fonctionnalités intégrées au langage.

Ce langage de programmation bénéficie d'une licence libre. Il fonctionne sur la plupart des plates-formes informatiques, des smartphones aux ordinateurs. Il est disponible pour tous ces systèmes d'exploitation (Windows, LINUX, Mac OS). Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

FIGURE 0.1 *Python 3.8*

5.5.2. L'environnement de programmation Pycharme

PyCharm est un environnement de développement intégré utilisé pour programmer en Python.
Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration du logiciel de gestion des versions, et supporte le développement web avec Django.Développé par l'entreprise tchèque JetBrains, c'est un logiciel multiplateforme qui fonctionne sous Windows, Mac OS X et Linux. Il est décliné en édition professionnelle.

FIGURE 0.2 *PyCharm community*

5.5.3. PyQt Bibliothèque

PyQt est un module libre qui permet de lier le langage Python avec la bibliothèque Qt distribué sous deux licences : une commerciale et la GNU GPL. Il permet ainsi de créer des interfaces graphiques en Python. Une extension de Qt Creator (utilitaire graphique de création d'interfaces Qt) permet de générer le code Python d'interfaces graphiques.

FIGURE 0.3 *Qt Bibliothèque*

5.3. Présentation de l'application

Pour mieux présenter notre approche, on a créé une interface qui donne la main à l'utilisateur d'introduire les paramètres de nos algorithmes.

Les taux de croisement et mutation sont fixées par défaut aux probabilités de 70% et 5%.

FIGURE 0.4 *La première interface*

- Les séquences de protéines sont extraites du benchmark « BaliBase » de format de fichier (.tfa) qui contient seulement les séquences protéines non-alignées

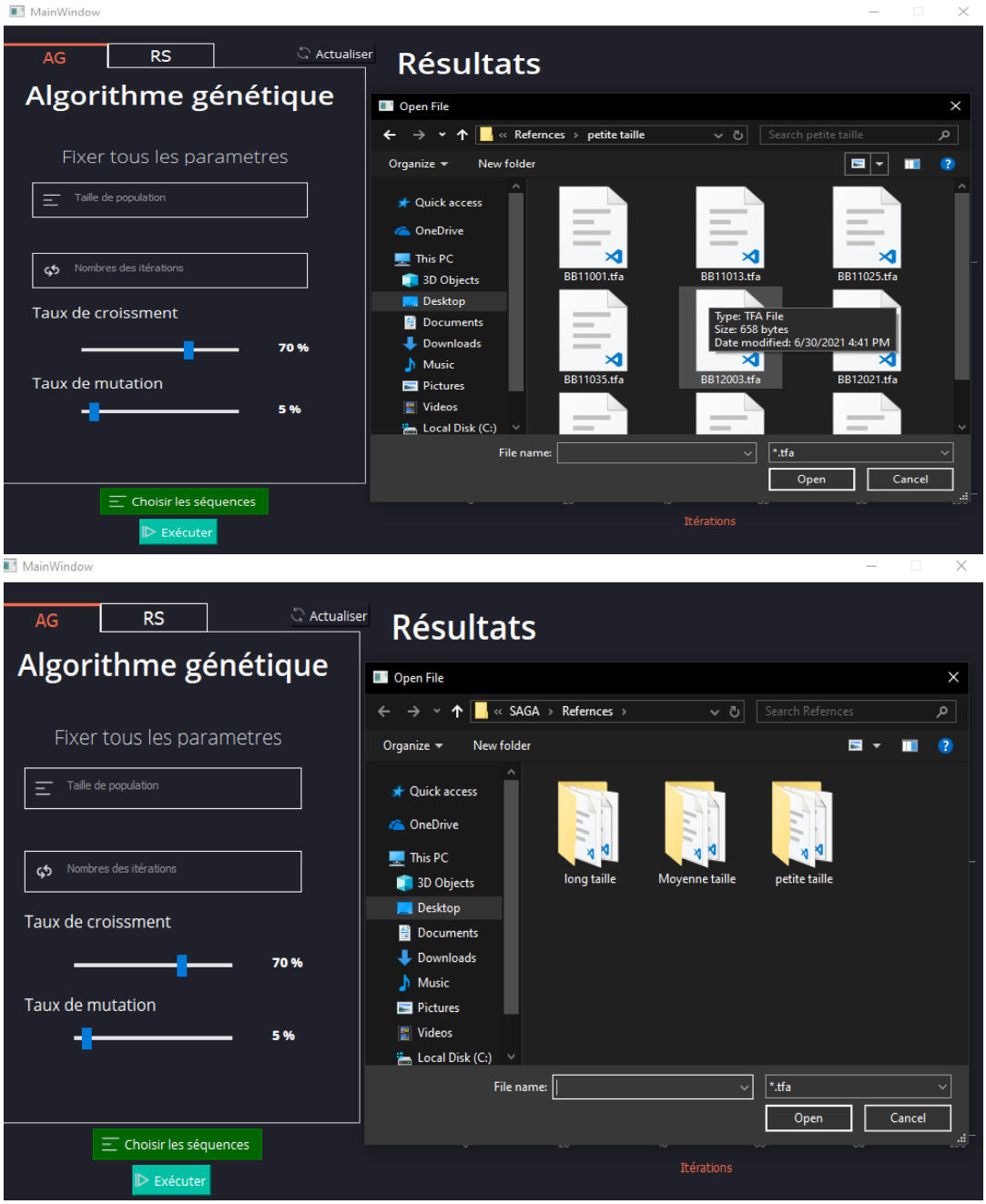

FIGURE 0.5 *Choisir un fichier de Benchmark "BaliBase" des séquences non-alignées*

- Pour Evaluer chaque alignement nous avons utilisant le principe de gap affine avec (**Pénalité d'ouverture d'un gap= 3/ pénalité d'extension d'un gap= 2)**

- Les résultats des deux approches (AG/AG+RS) sont vus dans les deux courbes côte à côte avec le temps d'exécution et le meilleur score de chaque approche.

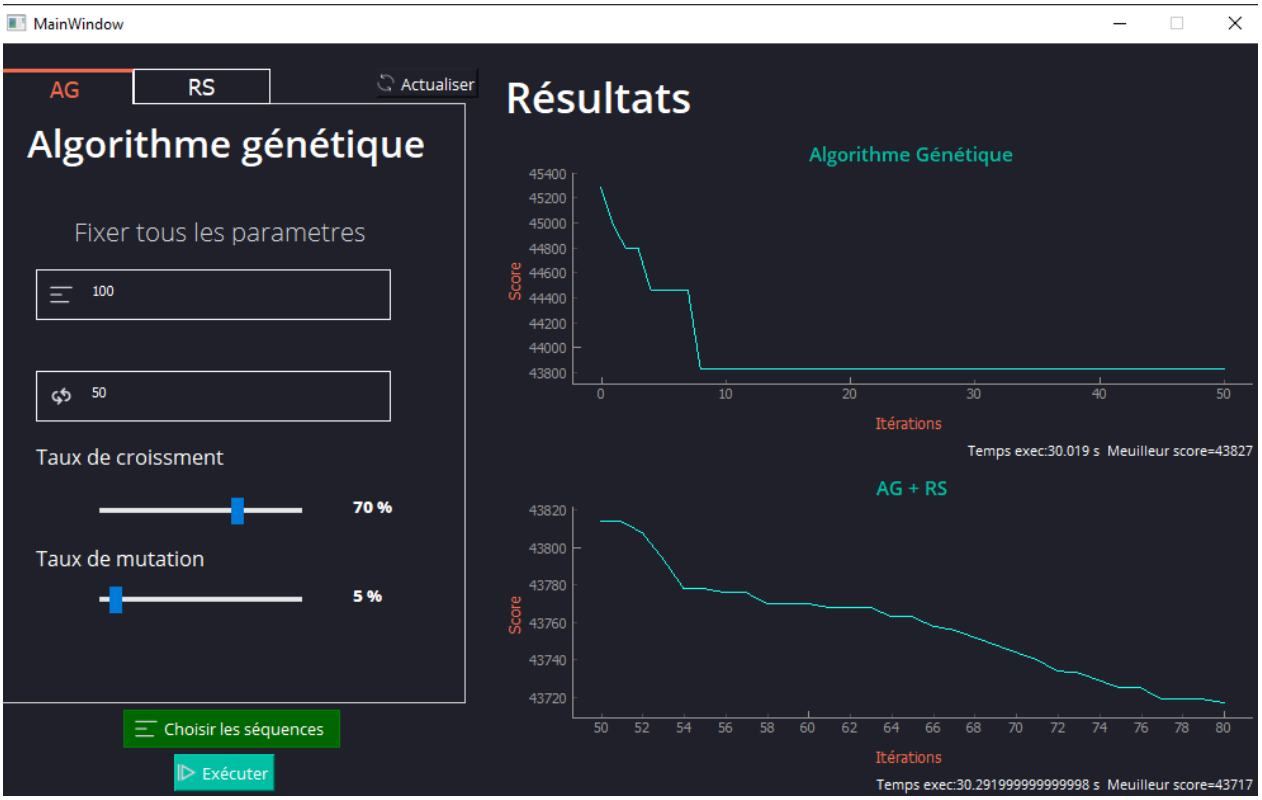

FIGURE 0.6 *Interfaces pour les résultats d'alignements*

5.4. Comparaison de Algorithme génétique avec Algorithme génétique Amélioré (AG+RS) :

Dans cette section, nous comparons les résultats entre les deux approches proposées à l'aide de l'opérateur GAP qui calcule l'écart entre les deux solutions, et qui est définit comme suit.

$$\mathrm{GAP}\left(\mathrm{AG}+\frac{\mathrm{RS}}{\mathrm{AG}}\right)=\frac{Score(AG)-Score(AG+RC)}{Score(AG)}X\ 100\%$$

Nous avons testé les résultats de notre application sur trois ensembles différents des instances tel que chaque type contient 5 instances, chaque instance contient entre 3 et 4 séquences, notre algorithme a été exécuté plusieurs fois pour chaque instance.

Les instances des séquences que nous avons utilisées sont extraite du benchmark « Balibase » qui est une base de données d'alignements de séquences multiples affinés manuellement, classés par blocs de base de longueur de séquence de conservation, de similarité et de présence d'insertions et d'extensions N/C-terminales.[50]

Les résultats sont présentés dans le tableau suivant. La première colonne représente le type de l'instance (Séquences avec la plus grande longueur/ Séquences de longueur moyenne /Séquences de petite longueur), la deuxième colonne et la troisième contiennent les résultats de l'approche AG et AG + RS, la quatrième colonne représente l'écart trouvé entre les solutions (GAP) et la dernière colonne contient la différence en temps d'exécution entre les deux approches.

Tous les résultats sont exécutés avec les mêmes paramètres : Température :50° /Seuil : 30°/ Nombre d'itération 20/ Taille de population 20 / Taux de croisement 70%/ Taux de mutation 5%.

Instance		AG		AG+RC		GAP (%)	Diff Temps Exécution (s)
		Temps exécution(s)	**Score**	**Temps exécution(s)**	**Score**		
Petite Taille	Instance1	0.146	282	0.154	278	**1.418**	**0.008**
	Instance2	0.112	973	0.119	943	**3.083**	**0.007**
	Instance3	0.064	443	0.068	436	**1.58**	**0.004**
	Instance4	0.152	1372	0.161	1351	**1.53**	**0.009**
	Instance5	0.284	2400	0.301	2384	**0.666**	**0.017**
Moyenne		**0.1516**		**0.1606**		**1.6554**	**0.009**
Moyenne taille	Instance6	0.565	8894	0.604	8820	**0.832**	**0.39**
	Instance7	1.035	7498	1.097	7481	**0.226**	**0.062**
	Instance8	2.278	17870	2.414	17824	**0.257**	**0.136**
	Instance9	1.058	7325	1.125	7293	**0.436**	**0.067**
	Instance10	0.741	17300	0.791	17183	**0.676**	**0.05**
Moyenne		**1.1354**		**1.2062**		**0.4854**	**0.0708**
Long taille	Instance11	7.187	52410	7.562	52349	**0.116**	**0.375**
	Instance12	3.53	70085	3.779	69975	**0.1569**	**0.249**

	Instance13	2.307	45024	2.457	44951	**0.162**	**0.15**
	Instance14	8.028	105833	8.569	105565	**0.253**	**0.541**
	Instance15	5.532	48899	5.94	48799	**0.204**	**0.408**
Moyenne		**5.3168**		**5.6614**		**0.15518**	**0.3446**
Totale						**0.7653**	**0.1414**

Table 0.1 *Tableau de comparaison de résultats d'alignement d 15références en utilisant les deux approches.*

D'après la Table 5.1 nous remarquons que :

- Dans le premier type d'instances (petites instances) : l'approche AG+RS a amélioré les résultats de l'AG par un pourcentage de 1.65%. Tandis que pour le temps d'exécution, la différence est négligeable, cela dû au fait que l'espace de recherche est petit.

- Dans le deuxième type d'instances (moyennes instances) : l'approche AG+RS a amélioré les résultats de l'AG avec un pourcentage de 0.48%. La différence dans le temps d'exécution est très petite (AG plus rapide que l'AG+ RS avec 0.06 s).

- Dans le troisième type d'instances (grandes instances) : l'approche AG+RS a amélioré AG avec un pourcentage de 0.15%. Alors que pour le temps d'exécution AG dépasse AG+RS avec en moyenne 0.33 s.

Nous pouvons ainsi conclure que :

- L'approche AG+RS a amélioré l'approche AG sur toutes les instances testées avec une moyenne de 0.8%.

- La différence dans le temps d'exécution des deux approches est insignifiante (~ 0.15 s).
- L'approche AG+RS peut être une méthode efficace pour résoudre le problème d'alignement multiple.

5.5. Conclusion

Dans ce chapitre, nous avons présenté l'environnent de développement et le langage de programmation utilisés, puis nous avons présenté les données de test utilisés et les résultats obtenus lors de l'application. Nous avons comparé l'approche AG proposé avec l'approche AG+RS amélioré. Les résultats ont montré l'efficacité de cette approche pour résoudre le problème d'alignement multiple.

Conclusion Générale

Conclusion Générale

Dans le cadre de ce travail de mémoire, nous avons traité un problème très important en bio-informatique qui est l'alignement multiple de séquences (MSA).

Au début, nous avons présenté quelques notions de base sur l'optimisation combinatoire (classes de problèmes, complexité, méthodes de résolutions …etc.).

Dans le deuxième chapitre, nous avons introduit quelques notions de base sur la bio-informatique nécessaires pour la compréhension de notre problème. Ce dernier a été détaillé dans le troisième chapitre qui s'est conclue avec un état de l'art sur le problème MSA.

Le troisième et quatrième chapitre étaient consacrés à la partie pratique. Où nous avons présenté notre approche proposée et les résultats obtenus. L'approche proposée est un algorithme génétique amélioré. L'amélioration est effectuée en appliquant l'algorithme de recuit simulé sur la meilleure solution trouvée par l'AG. L'expérimentation a montré l'efficacité de l'approche proposée.

Partie Bibliographique

Références

1. S. B. Needleman & C. D. Wunsch. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol.443–453.

2. T. F. Smith & M. S. Waterman. (1981). Identification of common molecular subsequences. J. Mol. Biol. 195–197.

3. Anbarasu LA, Narayanasamy P & Sundararajan V (2000) Multiple molecular sequence alignment by island parallel genetic algorithm.Curr Sci.78:858–863.

4. J. D. Thompson, F. Plewniak, & O. Poch. (1999). A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Res. 2682–2690

5. L. Wang &T. Jiang. (1994). On the Complexity of Multiple Sequence Alignment. J. Comput. Biol. 1(4), 337–348.

6. L. Abdesslem, « Utilisation des Approches d'Optimisation Combinatoire pour la Vérification des Applications Temps Réel », Université Mentouri de Constantine, 2010. [En ligne]. Disponible sur : https://bu.umc.edu.dz/theses/informatique/LAY5771.pdf

7. S. Christine, « Résolution de problèmes combinatoires et optimisation par colonies de fourmis ». [En ligne]. Disponible sur : https://perso.liris.cnrs.fr/christine.solnon/publications/poly.pdf

8. L. Kherbouche et Z. Oubahri, « Quelques méthodes de résolutions en optimisation combinatoire ». oct. 2017. [En ligne]. Disponible sur : https://dl.ummto.dz/bitstream/handle/ummto/2922/Kherbouche%2C%20Lynda.pdf?sequence=1&isAllowed=y

9. « Présentation des problèmes d'optimisation combinatoire ». https://www.emse.fr/~delorme/Papiers/MemoireDEA/memoire003.html (consulté le mai 01, 2021).

10. Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller et Edward Teller. Equation of State Calculations by Fast Computing Machines. The Journal of Chemical Physics, vol. 21, no. 6, pages 1087-1092, 1953. (Cité enpage82.).

11. « 41_ift1575_optimisation combinatoire-1. pdf ». Consulté le : mai 01, 2021. [En ligne]. Disponible sur : http://www.cours-examens.org/images/An_2013_2/Etude_superieure/Ingeniorat_informatique/ROMontreal/41_IFT1575_OptimisationCombinatoire-1.pdf

12. J Brownlee. Clever Algorithms: Nature-Inspired Program-ming Recipes. Lulu Enterprises Incorp orated, 2011. (Citéenpage82.)

13. I. AHMIA, « Une nouvelle métaheuristique pour les problèmes d'optimisation combinatoire : La Monarchie Métaheuristique », p. 106.

14. « Algorithme Génétique ». [En ligne]. Disponible sur : http://igm.univ-mlv.fr/~dr/XPOSE2013/tleroux_genetic_algorithm/fonctionnement.html

15. R. Jean-Loup, Bio-informatique et bio-analyse. [En ligne]. Disponible sur : http://excerpts.numilog.com/books/9782759208708.pdf

16. « A) Introduction générale ». 2000.

17. B. Nadira, « Optimisation Multi-Objectif Pour l'Alignement Multiple de Séquences ». 2007. [En ligne]. Disponible sur : https://bu.umc.edu.dz/theses/informatique/BEN4861.pdf

18. « Les changements chromosomiques à grande échelle ». [En ligne]. Disponible sur : http://www8.umoncton.ca/umcm-filion_martin/cours/genetique/Chapitre%2017.pdf

19. K. DJERBOUAI, « Alignement multiple des séquences protéiques par l'algorithme de recherche tabou ». 2018 2017.

20. A. LAYEB, « Approche quantique évolutionnaire pour l'alignement multiple de séquences en bio-informatique », Université Mentouri de Constantine Faculté des Sciences de l'Ingénieur Département d'Informatique, 2005. [En ligne]. Disponible sur : http://193.194.84.143/bitstream/handle/123456789/6788/LAY4382.pdf?sequence=1&isAllowed=y

21. D. Vincent, « Heuristiques pour la résolution du problème d'alignement multiple », l'Université d'Angers, 2008. [En ligne]. Disponible sur : https://tel.archives-ouvertes.fr/tel-00352784/document

22. V. Derrien, J.-M. Richer, et J.-K. Hao, « Plasma, un nouvel algorithme progressif pour l'alignement multiple de séquences », juin 2005.

23. « Vincent - 2008 - Heuristiques pour la résolution du problème d'al.pdf ». Disponible sur https://www.researchgate.net/publication/280750899_Plasma_un_nouvel_algorithme_progressif_pour_l%27alignement_multiple_de_sequences.

24. Fred Glover et Manuel Laguna.Tabu Search.Kluwer Academic Publishers, Norwell, MA, USA,1997. (Citéenpage17.)

25. Helena R Lourenço, Olivier C Martin et Thomas Stützle. Iterated local search. In Handbook of Metaheuristics, volume57of International Seriesin Operations Research and Management Science, pages321-353.KluwerAcademicPublishers,2002. (Citéenpage17.)

26. Helena R Lourenço, Olivier C Martinet Thomas Stützle. Iterated Local Search: Framework and Applications Handbook of Meta-heuristics. volume146ofInternationalSeriesinOperationsResearch&Management Science, chapitre12, pages363-397.SpringerUS, Boston, MA,2010. (Cité en page17).

27. N Mladenovi¢ et P Hansen. Variable neighborhood search. Computers & Operations Research, vol. 24, no. 11, pages10971100,1997. (Cité en page17)

28. S J Russell et P Norvig. Artifcial Intelligence : A Modern Approach. Prentice Hall,3rd édition,2009. (Citéenpage17)

29. [24Kirkpatrick1983a] S Kirkpatrick, C D Gelatt et M P Vecchi. Optimization by simulated annealing. Science, vol.220, pages671-680,1983. (Citéenpages17et82.)

30. V cerný. Thermodynamical approach to the traveling salesman problem: An effcient simulation algorithm. Journal of Optimization Theory and Applications, vol.45, no.1, pages41-51,1985. (Citéenpages17et82.)

31. Thomas A Feo et Mauricio G C Resende. A probabilistic heuristic for a computationally dificult set covering problem. Operations Research Letters, vol.8, no.2, pages67-71,1989. (Cité en page17.)

32. Thomas A Feo et Mauricio G C Resende. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, vol.6, pages109133, 1995. (Citéenpage17.)

33. Alberto Colorni,Marco Dorigo et Vittorio Maniezzo. Distributed Optimization by Ant Colonies. In European Conference on Artficial Life, pages134-142,1991. (Citéenpage17.)

34. James Kennedy et Russell C. Eberhart. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, pages1942-1948,1995. (Citéenpage17.)

35. Melanie Mitchell. An introduction to genetic algorithms. MIT Press, Cambridge, MA, USA,1998. (Citéenpage17.)

36. Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller et Peter Nordin. Genetic programming: An introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA,1998. (Citéenpage17.)

37. XianshunChen, Yew-Soon Ong, Meng-Hiot Lim et KayChen Tan. A Multi-Facet Survey on Memetic Computation. IEEE Trans. Evolutionary Computation, vol.15, no.5, pages591-607,2011. (Citéenpage17.)

38. Rainer Storn et Kenneth Price. Differential Evolution & Ndash; A Simple and Effcient Heuristic for Global Optimization over Continuous Spaces. J. of Global Optimization, vol.11, no.4, pages341-359, Décembre1997. (Citéenpage17.)

39. Rafael Martí, Manuel Laguna et Fred Glover. Principles of scatter search. European Journal of OperationalResearch, vol. 169, no. 2, pages359-372, 2006. (Citéenpage17.)

40. Pablo Rabanal, Ismael Rodríguez et Fernando Rubio. Using River Formation Dynamics to Design Heuristic Algorithms. In Selim G.Akl, Cristian S. Calude, Michael J. Dinneen, Grzegorz Rozen-berg et Todd Wareham, editeurs, UC, volume4618 of Lecture Notesin Computer Science, pages163-177. Springer, 2007. (Citéenpage17.)

41. Duc Truong Pham, Marco Castellani et Le Thi Hoai An. Nature-Inspired Intelligent Optimisation Using the Bees Algorithm. T. Computational Collective Intelligence, vol. 13, pages38-69,2014. (Citéenpage17.)

42. Swagatam Das, Arijit Biswas, Sambarta Dasgupta et Ajith Abra-ham. Bacterial Foraging Optimization Algorithm: TheoreticalFoun-dations, Analysis, and Applications. In Ajith Abraham, Aboul-Ella Hassanien, Patrick Siarry et Andries Engelbrecht, editeurs, Foundations of Computational Intelligence Volume 3SE-2, volume 203 of Studies in Computational Intelligence, pages 23-55. Springer Berlin Heidelberg, 2009. (Citéen page17.)

43. Xin-SheYang. Nature-inspired metaheuristic algorithms: Secondedition. LuniverPress, 2010. (Citéenpage17.)

44. Xin-She Yang et Suash Deb. Cuckoo Search via Lévy Flights. In NaBIC, pages210-214. IEEE. (Citéenpages17,49et50.)

45. Hamed Shah; Hosseini. The Intelligent Water Drops Algorithm a Nature Inspired Swarm; Based Optimization Algorithm. Int. J. Bio-Inspired Comput., vol.1, no.1/2, pages71-79, Janvier2009. (Citéenpage17.)

46. L.N. de Castro e tF. J. Von Zuben. Learning and Optimization Using the Clonal Selection Principle. Trans. Evol. Comp, vol. 6, no. 3, pages 239-251, Juin2002. (Citéenpage17.)

47. Zong Woo Geem, JoongHoon Kim et G V Loganathan. A new heuristic optimization algorithm: harmony search. Simulation, vol. 76, no. 2, pages 60-68, 2001. (Citéenpages17et53.)

48. Stefan Bottcher et AllonG. Percus. Extremal Optimization: Methods derived from Co-Evolution. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, MaxH. Garzon, Vasant Honavar, Mark Jakiela et Robert E. Smith, editeurs, Pro ceedings of the Genetic and Evolutionary Computation Conference, volume1, pages 825 832, Or-lando, Florida, USA,13-17 July 1999. Morgan Kaufmann. (Citéenpage17.)

49. Chris Voudouris et Edward Tsang. Guided Local Search. Rapport technique, European Journal of Operational Research, 1995. (Citéenpage18.).

50. Thompson JD, Plewniak F, Poch O. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. Bioinformatics. 1999 Jan;15(1):87-8. doi: 10.1093/bioinformatics/15.1.87. PMID: 10068696.

Annexes

Annexe1 : Méthode d ‘alignement en utilisant AG+RS.

```
#methode de calcul l'aligment en utilsant le Recuit simule
def AG(seuil,t,meuill):
    begin=time.clock()
    bestscore=[]
    i=0
    iterat=[]
    Scour=Solution(meuill)
    Scour.calculate_score()
    while t>=seuil:

        Si=copy.deepcopy(Scour)
        #Creer nouvelle voisin de la solution courante
        Si.voisinage()
        Si.calculate_score()
        operator_choice=random.random()
        #Condition ou la nouvelle solution est mieux que la courante
        if Si.score<=Scour.score :
            print("voisinage operator")
            Scour.sequence=Si.sequence
            Scour.calculate_score()
       #Cas ou la nouvelle solution est pire que la courante
        else:
            print("pass to next generation")
        #Ajouter meuilleur solution
        bestscore.append(Scour.score)
        iterat.append(t)
        t-=1
        i+=1
    return (bestscore,iterat,time.clock()-begin)
```

Annexe 2 : Méthode de sélection par roulette.

```python
def roulette_wheel_selection(rng: np.random.Generator,
                             population: np.ndarray,
                             fitness: np.ndarray,
                             size: int,indice) -> np.ndarray:
    if size > len(population):
        raise ValueError
    fitness_cumsum = fitness.cumsum()   # the "roulette wheel"

    fitness_sum = fitness_cumsum[-1]    # sum of all fitness values (size of the wheel)
    sampled_values = rng.random(size) * fitness_sum

    # For each sampled value, get the corresponding roulette wheel slot
    selected = np.searchsorted(fitness_cumsum, sampled_values)
    #print_report(population, fitness, fitness_cumsum, sampled_values, selected)
    return selected
```

Annexe 3 : Méthode d ‘alignement en utilisant l’AG.

```python
#methode Alignment en utilsant l'algorithme genetique
def AG(Nb_Iter,original_sequences,crois,mut,pop):
    begin=time.clock()
    max_seq=max(original_sequences,key=len)
    max_lent=len(max_seq)+2
    solutionss=pop
    bestalign=[]
    bestscore=[]
    scores_init=[]
    #evaluation de primiere population
    for sol in solutionss:
            sol.calculate_score()
            scores_init.append(sol.score) #remplir list des scores
    bestscore.append(min(scores_init))
    index=scores_init.index(bestscore[-1])
    bestalign.append(solutionss[index].sequence)
    #Parcourir les iterations
    for j in range(Nb_Iter):
        #list des scores
        scores=[]
        solutions=copy.copy(solutionss)
        i=0
        #Evaluation de generation
        for sol in solutions:
            sol.calculate_score()
            scores.append(sol.score) #remplir list des scores
            i+=1
        muttrv=False
        croistrv=False
        mini=min(scores)
        indice=scores.index(mini)
        select_numbers=int(len(solutions)/2)#numer de selection des individus 50%
        rng=default_rng()
        population_nb=np.arange(len(solutions))
        fitness=np.array(scores)
        #selection des individus en utilsant la methode de roulette
        select_individulas=roulette_wheel_selection(rng, population_nb, fitness,
select_numbers,indice)#selection par roullette

        while indice in select_individulas:
            select_individulas=roulette_wheel_selection(rng, population_nb, fitness,
select_numbers,indice)#selection par roullette
        operator_choice=random.random()
        #Condioton de faire l'opearation de mutation
        if operator_choice<mut:
            muttrv=True
            print("mutation operator")
            for i in range(len(select_individulas)):
                solutions[select_individulas[i]].mutation()
                solutions[select_individulas[i]].calculate_score()
        #Condioton de faire l'opearation de croissment
        if operator_choice<crois:
            croistrv=True
            print("croosover operator")
            for i in range(0,len(select_individulas),2):
                position=random.randint(2,max_lent)
                if i==len(select_individulas)-1:
                    results=
(solutions[select_individulas[i]].crossover(position,solutions[select_individulas[0]].sequence))
                    solutions[select_individulas[i]].sequence=results[0]
                    solutions[select_individulas[0]].sequence=results[1]
                    solutions[select_individulas[i]].calculate_score()
                    solutions[select_individulas[0]].calculate_score()
                else:
                    results=
(solutions[select_individulas[i]].crossover(position,solutions[select_individulas[i+1]].sequence))
                    solutions[select_individulas[i]].sequence=results[0]
                    solutions[select_individulas[i+1]].sequence=results[1]
                    solutions[select_individulas[i]].calculate_score()
                    solutions[select_individulas[i+1]].calculate_score()
        #Condition de pass au gen suivante
        if not muttrv and not croistrv:
            print("pass to next generation")
        scores=[]
        for sol in solutions:
            scores.append(sol.score) #remplir list des scores
        #Ajouter la meuilleur solution
        bestscore.append(min(scores))
        index=scores.index(bestscore[-1])
        bestalign.append(solutions[index].sequence)
    iterat=[]
    for i in range(Nb_Iter+1):
        iterat.append(i)
    minimu=max(bestscore)
    ind=0
    for i in range(len(bestscore)):
        if bestscore[i]<=minimu:
            minimu=bestscore[i]
            ind=i
    meuilleur=bestalign[ind]
    return (bestscore,iterat,time.clock()-begin,meuilleur)
```

Annexe 4 : Méthode calcule de score de deux solutions.

```
  #methode de calculer le score entre deux sequences
    def score_pairwise(self,seq1, seq2, gap_s, gap_e, gap = True):
        sum=0
        for A,B in zip(seq1, seq2):
            diag = ('-'==A) or ('-'==B)
            if diag:
                if gap:
                    sum+=gap_s
                else:
                    sum+=gap_e
            elif A!=B:
                sum+=1
        return sum
    #methode calculer score d'un solution
    def calculate_score(self):
        sc=0
        for i in range(1,len(self.sequence)):
            for j in range(0,i-1):
                sc+=self.score_pairwise(self.sequence[i], self.sequence[j], +3,+2)
        self.score=sc
        #print(self.score)
```

Annexe 5 : Croissement de deux solutions.

```
def signle_cross(self,seq1,seq2,ran,index):
        s1part1=seq1[0:ran]
        s1part2=seq1[ran:]
        s2part1=seq2[0:ran]
        s2part2=seq2[ran:]
        seq1=str(s1part1)+str(s2part2)
        seq2=str(s2part1)+str(s1part2)
        return (seq1,seq2)
    #croissmsnt avec autre solution
    def crossover(self,ran,seq):
        result1=[]
        result2=[]
        for i in range(len(self.sequence)):
            result1.append(self.signle_cross(self.sequence[i],seq[i],ran,i)[0])
            result2.append(self.signle_cross(self.sequence[i],seq[i],ran,i)[1])
        return (result1,result2)
```

Annexe 6 : Mutation d'une solution.

```
#methode de mutation
    def mutation(self):
        seqnb=random.randint(0,len(self.sequence)-1)
        s=list(self.sequence[seqnb])
        position=random.randint(0,len(self.sequence[seqnb])-1)
        if position!=len(self.sequence[seqnb])-1:
            s[position],s[position+1]=s[position+1],s[position]
        else:
            s[position],s[0]=s[0],s[position]
        self.sequence[seqnb]="".join(s)
```

Annexe 7 : Voisinage d'une solution.

```
def voisinage(self):
        seqnb=random.randint(0,len(self.sequence)-1)
        s=list(self.sequence[seqnb])
        position=random.randint(0,len(self.sequence[seqnb])-1)
        random_caracter = random.choice(acid_amines)
        s[position]=str(random_caracter)
        self.sequence[seqnb]="".join(s)
```