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 : SIOD17/M2/2019

Mémoire

présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Système d'Information Optimisation Décision**

Optimisation par algorithme génétique pour la résolution du problème d'assemblage de fragments d'ADN

Par :

BOUDOUH NOUARA

Soutenu le 06 juillet 2019, devant le jury composé de :

TOUIL Keltoum	MAA	Président
DJEROU Leila	Professeur	Rapporteur
OUAAR Hanane	MCB	Examinateur

Remerciement

Je tiens premièrement à remercier ALLAH le tout puissant de m'avoir donné le courage et la patience pour terminer ce modeste travail.

*Je tiens tous d'abord à remercier mon encadreur le professeur « **Djerou Leila »** pour sa patience, ses encouragements, ses conseils, et le temps précieux quelle ma accordé pour l'achèvement de ce travaille.*

Mes grands remerciements à tous les enseignants et collègues pour leur accompagnement, aide, encouragement et soutient tout le long de la réalisation de ce travaille.

Sans oublier tous les membres de ma famille pour leurs compréhensions, leurs soutiens, et leurs encouragements.

Dédicace

Je dédie ce modeste travail à :

La mémoire de ma mère.

Mon père.

Mon époux et mes enfants : Soumia, Khadidja, yahia et ma petite Asma.

A ma belle-sœur Akila.

A mes sœurs, mon frère Said et leurs familles, ainsi qu'à toute ma famille et mes amies.

Résumé

Le problème d'assemblage de fragments consiste à construire une séquence d'ADN à partir de plusieurs centaines (voire des milliers) de fragments obtenus par les biologistes du laboratoire. Il s'agit donc d'un problème d'optimisation combinatoire, connu comme étant NP-difficile. Pour résoudre ce problème, nous avons adopté l'algorithme génétique (AG). Pour améliorer la recherche de bonne solution, nous avons intégré l'algorithme PALS (Problem Aware Local search) comme un opérateur de recherche additionnel dans l'AG. Une étude expérimentale a montré que l'hybridation de l'AG avec PALS a donné de bons résultats par rapport à ceux de l'AG.

Table des matières

Table des matières

Liste des figures

Liste des algorithmes

Liste des tableaux

Introduction générale

La première étape dans l'étude du génome d'un organisme spécifique est la détermination de la séquence complète d'ADN : c'est le processus de séquençage d'ADN. C'est une étape cruciale dans tout projet de génomique, car elle conditionne fortement les succès des tâches postérieures comme l'identification des régions codantes et la prédiction des gènes.

Le séquençage Shotgun est la stratégie la plus utilisée dans les projets de séquençage de génomes [1]. Cette stratégie crée d'abord plusieurs copies de l'ADN initial. Puis, elle coupe de façon aléatoire chaque copie en plusieurs fragments, suffisamment courts pour être séquencés par un séquenceur. Ensuite, tous les fragments séquencés (ou lectures) sont assemblés l'un avec l'autre pour construire la séquence parent initiale. Cependant, après le découpage de la séquence initiale du génome en un nombre très grand de fragments, la position et la copie de génome de chaque fragment sont oubliées. De plus, du fait qu'un brin d'ADN peut être lu dans les deux sens, on ne connait pas aussi l'orientation du fragment. La détermination de l'ordre et de l'orientation de chaque fragment conduit au problème connu dans la bio-informatique, sous le nom de problème d'assemblage de fragments d'ADN.

Le problème d'assemblage de fragments d'ADN est un problème de rechercher un ordre total des fragments donnés qui aboutit à une séquence consensus reflétant avec précision la séquencé parent. Si la séquence parent est préalablement connue, on peut juger la qualité d'un résultat d'assemblage par une mesure de ressemblance entre la séquence consensus finale et la séquence parent connue. Cependant, en pratique, l'ensemble de fragments d'ADN est généré pour trouver la séquence parent.

La détermination de séquence d'ADN par l'assemblage de plusieurs centaines (voire des milliers) de ses fragments obtenus par les biologistes du laboratoire, est devenue une pratique indispensable dans les domaines de médecine, de phylogénétique et du crime (police technique). Il permet par exemple d'identifier un suspect sur la scène de crime, retrouver les liens de parenté entre deux personnes, établir des relations fonctionnelles entre les virus, les symptômes et les maladies, etc.

Dans la littérature [1], [2], le problème d'assemblage de fragments (FAP) est classé comme un problème d'optimisation NP-difficile ; pour n fragments, il y a $2^n n!$ configurations possibles, ou

2^n est le nombre de toutes les combinaisons possibles en termes d'orientation de fragments, et $n!$ est le nombre de toutes les permutations possibles des fragments.

Pour résoudre ce problème en un temps raisonnable, le recours aux méta-heuristiques d'optimisation est indispensable [2].

Dans ce projet, nous avons adopté l'algorithme génétique (AG) pour résoudre le problème d'assemblage de fragments. Pour améliorer la recherche de bonne solution, nous avons intégré une heuristique connue sous non d'algorithme PALS (Problem Aware Local search) [3] comme un opérateur de recherche additionnel dans l'AG ; après que l'algorithme AG localise une bonne région de solutions dans l'espace de recherche (exploration), PALS permet l'amélioration (exploitation) de cette région via la recherche locale. Par cette hybridation de AG et PALS, nous avons obtenu des résultats intéressants.

Ce mémoire comporte quatre chapitres organisés de la manière suivante :

— le premier chapitre est consacré aux concepts et notions biologique, ainsi qu'à la description du problème d'assemblage d'ADN et ses méthodes de résolution

— Le deuxième chapitre décrit les problèmes d'optimisation, les approches de leur résolution, nous insistions sur la méta-heuristique algorithme génétique.

— Le troisième chapitre présente l'hybridation de l'algorithme génétique AG avec PALS et la description détaillée de l'algorithme hybride AGPALS.

— L'implémentation du l'algorithme hybride AGPALS et les expérimentations et les résultats sont abordés dans le quatrième.

Le mémoire est terminé par une conclusion générale avec les perspectives envisagées.

CHAPITRE 1 :

Problème d'assemblage de fragments ADN

1. Introduction

Le problème d'assemblage de fragments d'ADN (ou DNA Fragment Assembly Problem : DNA FAP) consiste à construire la séquence d'ADN à partir de plusieurs centaines (voire des milliers) de fragments obtenus par les biologistes en laboratoire. C'est un problème issu du domaine de la bioinformatique et représente une tâche importante dans tout projet sur le génome.

Dans ce chapitre, nous abordons le problème d'assemblage de fragments d'ADN, qui se pose au niveau du séquençage des génomes. Nous commençons le chapitre par la présentation des concepts d'analyse d'ADN puis nous décrirons le processus de séquençage ADN et enfin nous exposons les différentes méthodes et les techniques d'assemblage.

2. Analyse de l'ADN

L'ADN, abréviation d'acide désoxyribonucléique, se trouve dans le noyau de la plupart des types de cellules. Les cellules ayant un noyau et des compartiments sont appelées eucaryotes. Par contre, les bactéries n'ont pas de noyau et, pour cette raison, sont appelées procaryotes [4].

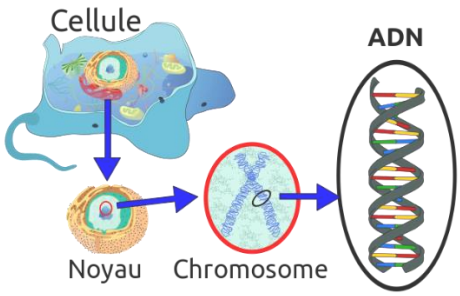


Figure 1.1 : Origine d'ADN

L'objectif principal de l'analyse d'ADN est de déterminer la séquence complète du génome et de définir son contenu génétique.

L'analyse d'ADN est devenue de nos jours une pratique indispensable dans les domaines de médecine, de phylogénétique et du crime (police technique), comme il est mentionné dans le paragraphe suivant :

« La preuve par l'ADN a créé un bouleversement important au sein des milieux scientifique et judiciaire, en faisant en sorte que l'identification génétique soit possible autrement que par des tests sanguins conventionnels. Les techniques d'analyse de l'ADN se sont peaufinées au

point de permettre l'obtention de résultats concluants à partir d'un infime échantillon d'ADN. Pour démontrer le haut degré de fiabilité des méthodes qu'ils utilisent, les laboratoires qui pratiquent des analyses judiciaires se font accréditer. Un des bénéfices de l'utilisation de la preuve par l'ADN est celui de pouvoir établir avec davantage de précision les liens de filiation entre individus apparentés. »[5]

Dans ce cas, l'analyse d'ADN permet de :

- Identifier des victimes, d'établir la présence de suspects sur le lieu du crime,
- Etablir des relations fonctionnelles entre les virus, les symptômes et les maladies, etc.
- Etudier les relations évolutives des espèces dans l'analyse phylogénétique.
- Classifier des espèces, en biologie.

2.1 Structure de l'ADN

L'ADN (pour acide désoxyribonucléique) est le support de l'hérédité, sa structure a été découverte en 1953 par Watson et Crick. Les molécules d'ADN sont les plus grosses molécules du monde vivant, elles sont présenteés dans tous les organismes vivants.

Une molécule d'ADN est une double hélice composée de deux brins enroulés l'un autour de l'autre, chacun de ces brins est constitué d'assemblage de plusieurs nucléotides accrochés les uns aux autres par des liaisons phosphodiester, dont l'ordre d'enchaînement est très précis et correspond à l'information génétique [6].

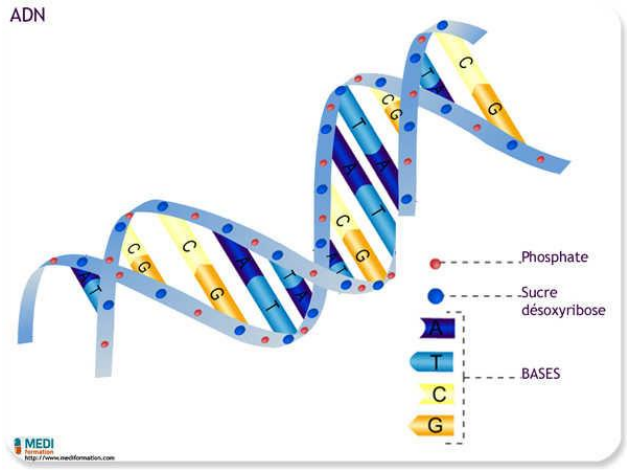


Figure 1.2 : Structure de l'ADN

Un nucléotide est constitué un sucre (désoxyribose), d'un acide phosphorique et d'une base azotée. 4 bases sont possibles : - Adénine - Thymine - Guanine – Cytosine[7].

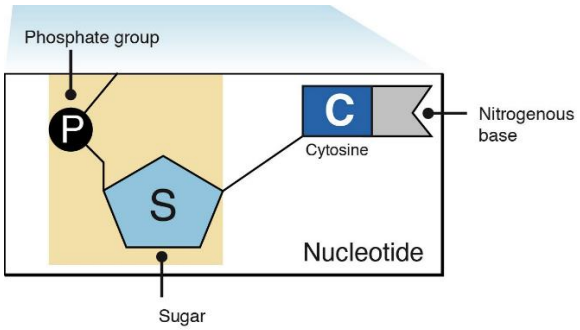


Figure 1.3 : Structure d'un nucléotide d'ADN

2.2. Les étapes d'analyse de l'ADN

L'extraction de l'ADN se fait à partir de la totalité ou d'une partie de l'organisme étudié, même si un grand nombre de cellules est utilisé pour réaliser l'extraction, la quantité finale d'ADN est généralement faible. Alors l'étape suivante consistera donc à amplifier cette quantité de l'ADN extraite, puisque la séquence de l'ADN est très longue (la taille du génome humain est approximativement de 3,2 milliards de paires de nucléotides), on doit la découper pour l'analyser. La figure suivante montre le processus d'analyse d'ADN [8].

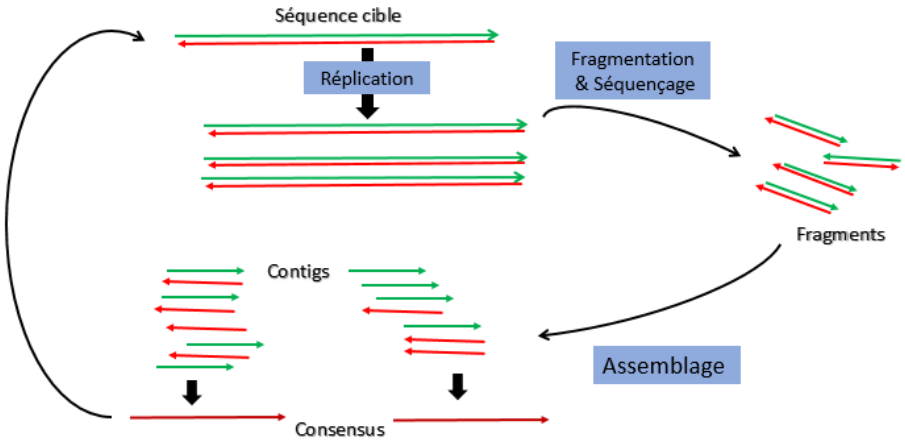


Figure 1.4 : Processus d'analyse d'ADN

2.2.1 Réplication de la séquence d'ADN

La réplication de la séquence d'ADN est l'opération de duplication ou d'amplification, elle est réalisée à partir d'une technique appelée PCR ou "Polymerase Chain Reaction" (réaction de polymérisation en chaîne). Cela permet de dupliquer à plusieurs millions d'exemplaires un fragment d'ADN grâce à l'ADN polymérase [3].

Une duplication est une mutation qui génère un dédoublement d'une partie de l'ADN génomique. La duplication peut recouvrir l'ensemble du génome (formation de polyploïdes), un chromosome entier, ou un fragment de chromosome de taille plus ou moins grande.

Les duplications peuvent éventuellement entraîner l'apparition de copies multiples d'un ou plusieurs gènes, provoquant ainsi une certaine redondance de l'information génétique [2].

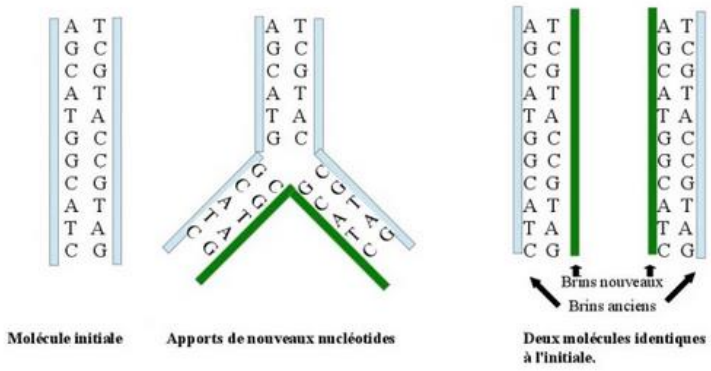


Figure 1.5 : Duplication de l'ADN

2.2.2. Fragmentation l'ADN

L'analyse d'une molécule d'un ADN de taille importante nécessite son découpage en fragments de taille compatible avec le séquençage. Deux différentes approches de fragmentation ont principalement été utilisées :

- **La fragmentation aléatoire** ; on découpe directement l'ensemble de l'ADN à séquencer en petits morceaux de taille optimisée pour le séquençage (1000 paires de bases) [9].

- **Segmentation après cartographie** ; dans le cas des génomes de très grande taille, la complexité de reconstruction devient une difficulté sérieuse, c'est pourquoi dans ces cas-là, certaines équipes ont eu recours à une approche à deux niveaux (segmentation après cartographie)[10].

3. Séquençage de fragments d'ADN.

La première étape dans l'étude du génome d'un organisme spécifique est la détermination de la séquence complète d'ADN : c'est le processus de séquençage d'ADN. Les informations fournies par cette étape servent à identifier les fonctions des gènes dans le génome.

Le séquençage d'ADN est le processus qui consiste à déterminer l'ordre d'enchaînement des nucléotides d'un fragment d'ADN donné. Il est réalisé par des machines de séquençage (des séquenceurs)[11].

Un séquenceur de gènes est un appareil capable d'automatiser l'opération de séquençage de l'ADN. Il peut déterminer la séquence des bases A, C, T et G, de gauche à droite le long d'un brin d'une double hélice d'ADN jusqu'à une longueur moyenne typique, et a un taux d'erreur typique. La courte séquence fournie par le séquenceur est appelé une lecture (issue du terme anglais read). La figure suivante montre des exemples de séquenceurs d'Illumina

Figure 1.6 : un séquenceur d'Illumina

Les capacités d'un séquenceur sont définies par :

- La longueur des lectures (reads) produits (L)
- Le nombre des lectures (reads) produits (n)
- Le nombre de nucléotides lu : (L x n)
- Le temps de séquençage
- La qualité du séquençage

3.1. Les méthodes de séquençage

Apres l'apparition de la technique de séquençage, différentes méthodes de séquençage de l'ADN ont été développées.

3.1.1 Séquençage de première génération

Le séquençage de l'ADN a été inventé dans la deuxième moitié des années 1970. Deux méthodes ont été développées indépendamment en 1977, l'une par l'équipe de Maxam et Gilbert aux États-Unis, et l'autre par Frederick Sanger en Grande-Bretagne. Ces deux méthodes sont fondées sur des principes diamétralement opposés ; l'approche de Sanger est

une méthode par synthèse enzymatique sélective, tandis que celle de Maxam et Gilbert est une méthode par dégradation chimique sélective.

A la fin des années 80, une automatisation de la méthode Sanger 80 a été réalisée, avec le développement des marquages fluorescents et de l'électrophorèse capillaire, qui a ouvert la voie du séquençage à haut débit.

La technique de séquençage avec des didésoxyribonucléotides (ddNTP) fluorescents ("dye terminator sequencing") utilise des didésoxyribonucléotides dont chacun est marqué par un fluorophore spécifique. Les fragments d'ADN synthétisés portent ce fluorophore terminal. On les appelle des terminateurs d'élongation ou "Big Dye Terminators" ou "Dye-labeled terminator"[12].

3.1.2 Séquençage de deuxième génération (NSG)

Le terme *NGS* « Next generation sequencing » regroupe l'ensemble des technologies ou plateformes de séquençage développées depuis 2005 par quelques sociétés de biotechnologies, l'émergence de cette nouvelle technologie est considérée comme la seconde génération de séquençage [13].

Avec ces technologies, on peut séquencer de grandes quantités d'ADN en des temps records, et la taille d'un fragment peut être entre 150 à 300 pb.

3.1.3 Les nouvelles technologies de séquençage à très haut débit (NGST)

Les améliorations technologiques ont rapidement changé le domaine du séquençage de l'ADN, une révolution en génomique fonctionnelle a eu lieu depuis 2012, avec l'avènement des technologies de séquençage à très haut débit NGST ("next-génération high throughput DNA sequencing technologies")[14].

Des quelques 800 à 1000 nucléotides qu'un chercheur pouvait espérer séquencer en quelques jours par des techniques lourdes, complexes et dangereuses (utilisation d'isotopes radioactifs) dans les années 80, on est arrivé à l'heure actuelle à des techniques de séquençage simplifiées qui génèrent des millions de nucléotides par jour. L'ensemble des données de séquençage est implémenté en temps réel dans des bases de données pour leur analyse.

3.2. Le séquençage de Shotgun.

Le séquençage de Shotgun est méthode de séquençage introduite en 1982 par Sanger. Cette methode commence par découper de façon aléatoire le génome en différentes sections de longueur prédéterminée : 2.000 paires de base, 10.000 paires de base ou 50.000 paires de base.

Des algorithmes mathématiques permettent ensuite d'assembler les fragments qui se suivent et de leur attribuer leur véritable emplacement sur le génome [15]. La méthode de séquençage de Sotgun comporte les étapes suivantes :

- Extraire l'ADN : L'extraction de l'ADN se fait à partir de la totalité ou d'une partie de l'organisme étudié. De nombreux protocoles existent,
- Découpage du génome en fragments qui sont insérés dans un vecteur (BAC, YAC, …).
- Clonage de l'ADN et séquençage des extrémités du clone
- Cosmide ou BAC sous-cloné en fragments de petite taille
- Des clones prélevés aléatoirement sont séquencés
- Recueil des données brutes du séquenceur
- Des séquences contigües sont reconstruites par recouvrement
- Assembler la séquence par création d'un graphe de DE Bruijn qui peut comporter plus de 100 millions de noed et dont les segment sont appelées K-mers.
- Finalisation de la séquence en comblant les tous : une couverture plus importante aboutit à des trous plus petits et moins nombreux
- Recherche de la signification phénotype et clinique via les polumorphisme vucléotides , insertion/suppression variant, variation du nombre d'exemplaire et mutations.

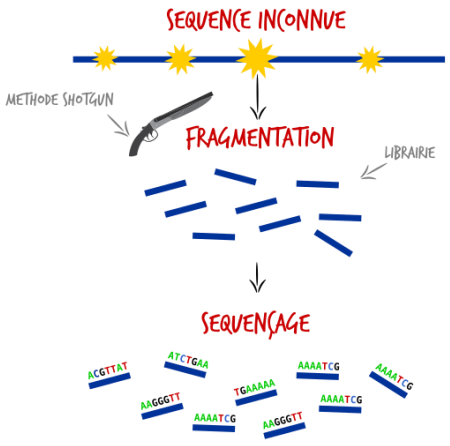


Figure 1.7 : Séquençage de Sotgun

4. L’assemblage de fragments d’ADN

L'assemblage de fragments d’ADN consiste à reconstituer (ou bien fusionner) des fragments d'ADN issus d'une plus longue séquence et produites par un séquenceur. Le programme informatique qui réalise cette tâche est appelé assembleur. Quelques termes sont employés lorsqu’on traite le problème d’assemblage de fragments d’ADN [16], comme :

- **Lecture** (read) : une séquence d'un fragment.
- **Contigs** : séquences continues générées par l'alignement de séquences de fragments qui se chevauchent.
- **Brèches** ("gaps") : parties du génome non séquencées ou dont les séquences ne chevauchent pas avec d'autres et ne peuvent donc entrer dans un contig.
- **Régions** de faible complexité : parties du génome dont les séquences sont très peu diversifiées comme les séquences répétées.

Il existe deux types d’assemblage :

4.1. Assemblage par cartographie (Mapping)

Son principe est de comparer les fragments issus de la phase de séquençage avec une séquence ADN déjà connue. Trouver une solution à ce problème revient donc à chercher, pour chacun des fragments la position qui marque un chevauchement maximal avec le génome de référence [16].

4.2. Assemblage de novo

L’assemblage de fragments ADN sans aucun génome de référence est appelé assemblage de novo. Il est utile dans le cas où l’ADN provient d’une source inconnue (par exemple les cheveux trouvés sur une scène de crime). Ce type d’assemblage se base sur la vérification des chevauchements deux à deux entre les fragments. Trouver la solution exacte est équivalent à effectuer un nombre de comparaison exponentiel par rapport au nombre de fragments. Ceci est impossible de l’effectuer dans un temps raisonnable.

Il existe un certain nombre des méthodes d’assemblage de novo, l’approche OLC **(**Overlap-Layout-Concensus) est l'une des premières méthodes utilisées pour assembler une séquence génomique avec succès.

4.3. L'approche OLC (Overlap-Layout-Consensus)

Le principe de l'approche OLC (Overlap-Layout-Consensus) est de fusionner des séquences chevauchantes, en commençant par celles ayant les scores les plus élevés, jusqu'à obtention d'une séquence unique [16].

Les assembleurs utilisant la méthode « overlap-layout-consensus » sont plutôt utilisés avec des lectures allant de 100 à 800 pb principalement avec un séquençage Sanger.

Le processus d'assemblage l'approche OLC se déroule en trois phases ou étapes [16].

Chevauchement (overlap)

Cette phase consiste à trouver les fragments qui se chevauchent. Elle consiste à chercher le bon ou le plus long alignement entre le suffixe d'une lecture et le préfixe d'une autre lecture. L'intuition derrière la recherche des chevauchements entre paires de lectures est que deux lectures suffisamment chevauchantes se retrouvent très probablement l'une à côté de l'autre dans la séquence cible.

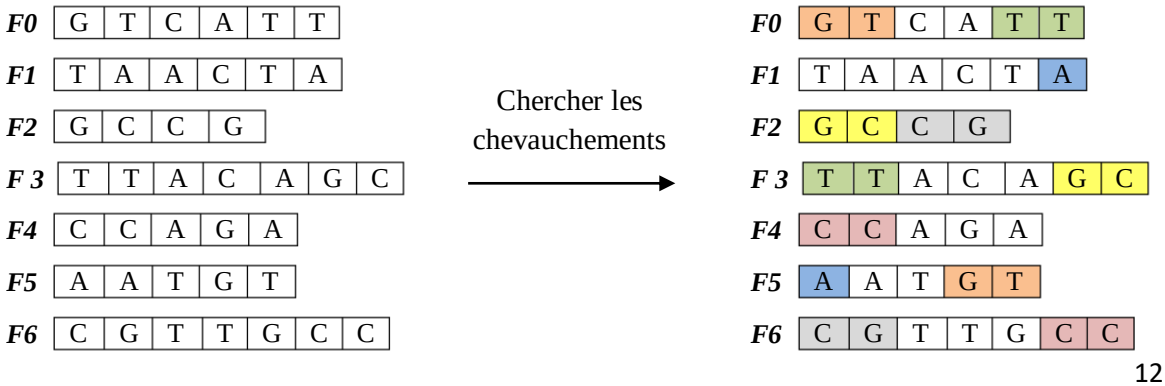
Alignement (layout)

Cette phase consiste à chercher un ordre plausible de fragments en se basant sur les chevauchements calculés. C'est la plus difficile étape parce que la décision d'assembler deux fragments se base sur leur chevauchement qui peut être approximatif du fait des erreurs de séquençage. Pour la plupart des assembleurs, des paires de lectures sont sélectionnées en fonction de leur score (les scores les plus élevés en premier), si l'alignement est considéré comme cohérent, les deux séquences sont fusionnées pour former un contig [16].

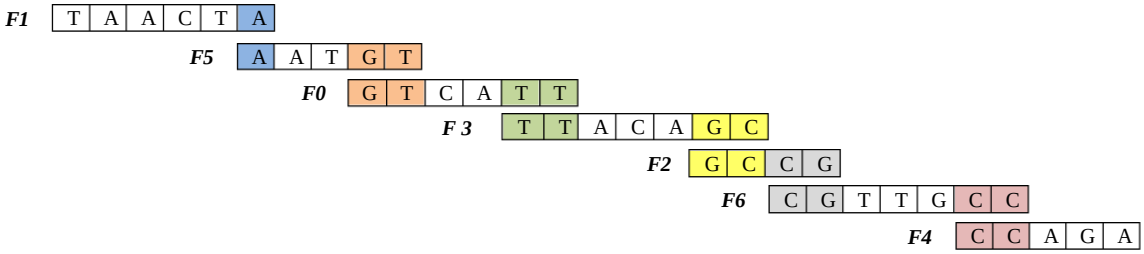
Consensus

Cette phase consiste à produire une séquence d'ADN à partir du résultat de la phase d'alignement.

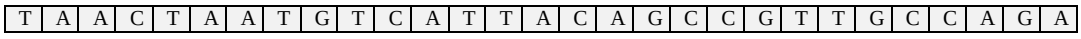
Un exemple illustratif des trois phases de l'approche OLC est donné dans la figure 1.8
.



1- Etape de chevauchement (overlap)



2- Etape d'alignement



3- Etape de Consensus

Figure 1.8 : Schéma illustratif de la méthode OLC (Overlap-Layout-Consensus)

5. Les problèmes d'assemblage

Le problème de l'assemblage peut être comparé à celui de la reconstruction du texte d'un livre à partir de plusieurs copies de celui-ci, préalablement déchiquetées en petits morceaux. Parmi les problèmes d'assemblage d'ADN :

- Toutes les machines de séquençage imposent une longueur de lecture maximale et produisent des erreurs.
- Orientation inconnue : une fois la séquence d'ADN d'origine est fragmentée, l'orientation des fragments est perdue et l'extrémité à sélectionner devient inconnue.
- Régions répétées : pour un assembleur deux lectures se chevauchant parfaitement proviennent de la même région génomique. Or, certaines séquences peuvent se retrouver de multiples fois dans un génome.
- Les assembleurs actuels ne gèrent pas d'une manière assez efficace les longues séquences répétées.
- Faible couverture ; la distribution de la couverture sert à mesurer la qualité de la séquence consensus. Plus la couverture est grande, moins il y a de gaps, et meilleur est le résultat. La couverture *C* est donnée par :

$$C = \frac{\sum_{f \in L} |f|}{G} \qquad (1.1)$$

Ou

- $\boldsymbol{L}$ est la collection de fragments d'ADN.
- $|\boldsymbol{f}|$ est la longueur du fragment $\boldsymbol{f} \epsilon \boldsymbol{L}$.
- $\boldsymbol{G}$ est la longueur de la séquence cible.

6. Conclusion

Dans ce chapitre, nous avons fait une étude détaillée sur le problème d'assemblage de fragments d'ADN. Nous avons présenté quelques méthodes de résolution du problème d'assemblage de fragments ADN qui se distinguent par la caractérisation des génomes séquencés, le développement des technologies de séquençage, et l'utilité du séquençage d'un génome spécifique.

Nous avons vu que le processus d'assemblage présente plusieurs difficultés qui rendent le problème d'assemblage de fragments d'ADN comme un problème difficile, d'où plusieurs heuristiques et méta-heuristiques sont proposés. Dans le chapitre suivant, nous allons exposer l'heuristique et le méta heuristique que nous allons utiliser, dans notre projet, pour résoudre ce problème.

CHAPITRE 2 :

Méta-heuristiques d'optimisation

1. Introduction

L'optimisation combinatoire (OC) occupe une place très importante en recherche opérationnelle et en informatique, c'est un outil essentiel pour le scientifique et pour l'ingénieur en particulier. Elle couvre un large éventail de techniques et fait toujours l'objet de recherches intensives. Ses domaines d'application sont extrêmement variés.

Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir, ils sont généralement difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données.

Les méta-heuristiques constituent une alternative très intéressante pour traiter les problèmes d'optimisation combinatoire, car elles sont constituées d'un ensemble de concepts fondamentaux (par exemple, la notion du mémoire et les mécanismes d'intensification et de diversification), qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation combinatoire. Parmi les méta-heuristiques les plus largement utilisées, nous retenons celle basée sur la théorie de sélection comme l'algorithme génétique.

Ce chapitre est consacré à un état de l'art des différentes approches couramment employées pour résoudre un problème d'optimisation combinatoire. Nous ferons en particulier une description de la méta-heuristique algorithme génétique (AG).

2. Problème d'optimisation

Un problème d'optimisation se définit comme la recherche, parmi un ensemble de solutions possibles S (appelé aussi espace de décision ou espace de recherche), de la solution(s) x^* qui rend minimale (ou maximale) une fonction mesurant la qualité de cette solution.

Cette fonction est appelée fonction objectif. Si l'on pose $f: S \to \mathbb{R}$, la fonction objectif à minimiser (respectivement à maximiser) à valeurs dans $\mathbb{R}$, le problème revient alors à trouver l'optimum $x^* \in S$ tel que $f(x^*)$ soit minimal (respectivement maximal).

Lorsque l'on veut résoudre un problème d'optimisation, on recherche la meilleure solution possible à ce problème, c'est-à-dire l'optimum global. Cependant, il peut exister des solutions intermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreint de l'espace de recherche : on parle alors d'optimums locaux. Cette notion est illustrée dans la figure 2.1.

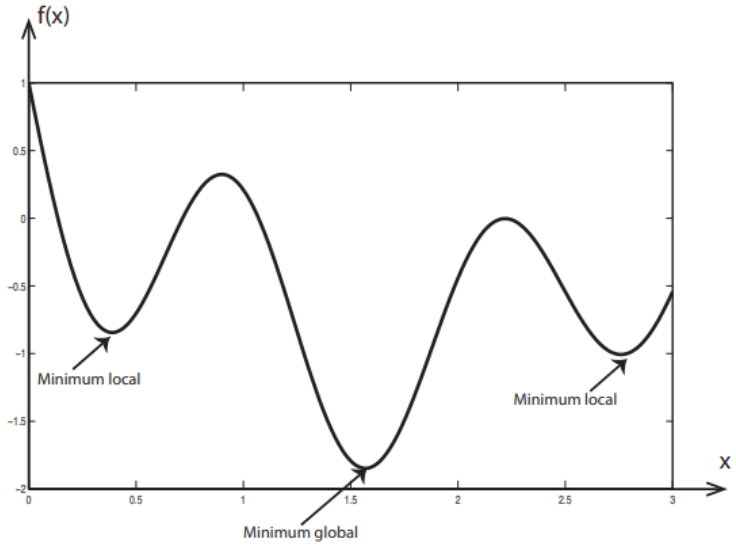


Figure 2.1 : Différence entre un optimum global et des optima locaux

Selon la nature des « espaces » dans lesquels les variables de décision prennent leurs valeurs, on peut faire une distinction entre l'optimisation continue de l'optimisation discrète (ou combinatoire). Pour l'optimisation continue, les solutions sont des vecteurs de réels (on parle de variables réelles), et l'espace de variable de décision (l'espace de recherche) est infini. Cependant pour l'optimisation discrète, l'espace de recherche est fini et discret. Les problèmes discrets sont généralement combinatoires.

Un problème d'optimisation combinatoire « POC » (on dit aussi d'**optimisation discrète**) est un problème d'optimisation dont les ensembles réalisables sont finis mais combinatoire. Un problème d'optimisation combinatoire peut être défini par :

– Un vecteur de variables $x = (x_1, x_2, ..., x_n)$,

– Domaine des variables $D = (D_1, D_2, ..., D_n)$, où les $(D_i)_{,i=1,...,n}$ sont des ensembles finis,

– Ensemble de contraintes,

– Une fonction objectif f à minimiser ou à maximiser,

– Ensemble de toutes les solutions réalisable possibles est $S = \{x = (x_1, x_2, ..., x_n) \in D$ / x satisfait toutes les contraintes$\}$, l'ensemble S est aussi appelé un espace de recherche.

La résolution d'un problème d'optimisation combinatoire consiste à :

- Modéliser un problème : espace de recherche, solutions
- Formaliser mathématiquement : fonction objectif, contraintes
- Appliquer d'une méthode d'optimisation
- Obtention d'une solution

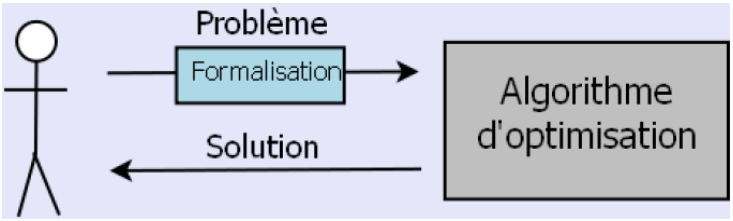


Figure 2.2. : Principe de résolution d'un POC

La résolution de (POC) consiste à trouver la meilleure solution, définie comme la solution globalement optimale ou un optimum global ou trouver dans un ensemble discret un parmi les *meilleurs* sous-ensembles (ou solutions) réalisables, la notion de *meilleure solution* étant définie par une fonction objectif.

3. Méthodes de résolution

Il existe plusieurs méthodes de résolution d'un problème d'optimisation combinatoire qui sont divisées en deux catégories : méthodes exactes et méthodes approchés

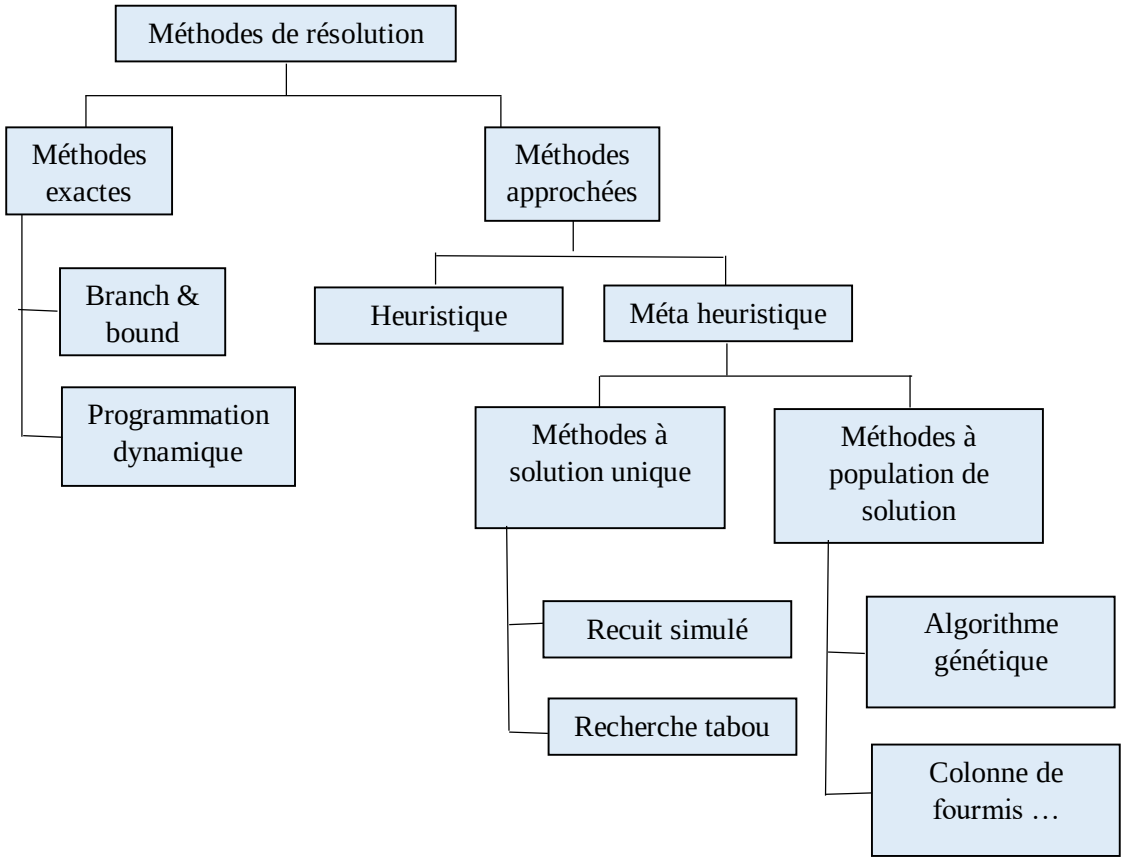


Figure 2.3 : Les méthodes de résolution d'un POC

3.1 Méthodes de résolution exactes

Les méthodes exactes (exhaustives) explorent l'espace de recherche dans sa totalité pour trouver des solutions optimales aux problèmes de taille raisonnable. Elles rencontrent généralement des difficultés face aux applications de taille importante, elles sont nombreuses et e caractérisent par le fait qu'elles permettent d'obtenir une ou plusieurs solutions dont l'optimalité est garantie.

3.2 Méthodes de résolution approchées

Les méthodes approchées (approximatives) ne garantissent pas de trouver une solution exacte, mais seulement une approximation. Elles explorent une sous-partie de l'espace de recherche. Les méthodes approchées peuvent être des heuristiques ou des méta-heuristiques, elles exploitent généralement des processus aléatoires dans l'exploration de l'espace de recherche.

3.2.1 L'heuristique

Généralement une heuristique est un algorithme approché (pas de garantie d'optimalité) conçue pour un problème particulier et ne peut pas être généralisée, en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée.

L'usage d'une heuristique est efficace pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte. Les heuristiques peuvent être classées en deux catégories : méthodes constructives et méthode de recherche locale

Méthodes constructives 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. L'algorithme montré dans la figure suivante, présente le principe d'une heuristique constructive.

```
Fonction HeuristiqueConstructive() : solution s
    s : solution,
    s ← ∅ ;

    Tant que ( s n'est pas une solution complete) faire
        choisir un nouvel element de solution e
        s ← s + e ;
    Fait
    Retourner s ;
Fin
```

Figure 2.4 : HeuristiqueConstructive

Méthode de recherche locale (ou de fouilles locales) démarrent avec une solution initialement complète s_0 (probablement moins intéressante), et de manière répétitive essaie d'améliorer cette solution en explorant son voisinage $v(s)$. L'algorithme suivant montre le principe d'une heuristique de recherche locale.

```
Fonction RechercheLocale(f, Ω, s0) : meilleure solution s*
    s* : solution, z : reel

    s* ← s0 ; z ← f(s*) ;

    Tant que (∃ s' dans ϑ(s*)  ET acceptable(s')) faire
        s* ← s' ;
        z ← f(s*) ;
    Fait
    Retourner s* ;
Fin
```

Figure 2.5 : RechercheLocale

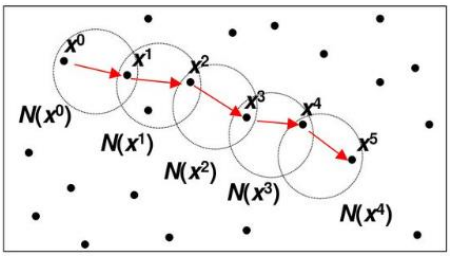


Figure2.6 : Illustration de méthode de recherche locale

L'algorithme PALS (Problem Aware Local Search) est une heuristique de recherche locale permettant de diversité la recherche pour trouver des solutions beaucoup plus précises. Cette heuristique a été utilisée comme opérateur de mutation dans un algorithme génétique (GA) [5].

3.2.2 Méthodes méta-heuristiques

Face aux difficultés rencontrées par les heuristiques pour avoir une solution réalisable de bonne qualité pour des problèmes d'optimisation difficiles, les méta-heuristiques ont fait leur apparition. Ces algorithmes sont plus complets et complexes qu'une simple heuristique, et permettent généralement d'obtenir une solution de très bonne qualité [6].

Les méta-heuristiques sont le plus souvent itératives, ainsi le même processus de recherche est répété lors de la résolution. Il existe deux approches des méthodes méta-heuristiques, méta-heuristiques à solution unique et méta-heuristiques à population de solutions.

Méta-heuristiques avec une solution unique sont appelées méthodes de trajectoire, contrairement aux méta-heuristiques à base de population, les méta-heuristiques à solution unique commencent avec une seule solution initiale et s'en éloignent progressivement, en construisant une trajectoire dans l'espace de recherche. Exemple : méthode du recuit simulé, la recherche tabou, … etc.

Méta-heuristiques avec une population de solutions

Parmi les méta-heuristiques, avec une population de solutions, les plus utilisées dans la littérature, on trouve l'algorithme génétique.

4. Algorithmes génétiques (AG)

Les algorithmes génétiques (AG) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Ils ont été adaptés à l'optimisation par

John Holland en 1975. Ils fournissent des solutions aux problèmes n'ayant pas de solutions calculables en temps raisonnable de façon analytique ou algorithmique.

Un algorithme génétique est défini par :

- Individu/chromosome/séquence : une solution potentielle du problème :
- Population : un ensemble de chromosomes ou de points de l'espace de recherche ;
- Environnement : l'espace de recherche ;
- Fonction de fitness : la fonction - positive - que nous cherchons à maximiser ou minimiser.

L'application de l'AG pour résoudre un problème d'optimisation nécessite la définition de :

- Une représentation sous la forme d'un chromosome des solutions (codage).
- Une fonction objectif pour évaluer la qualité, en termes de *fitness*, de chaque individu.
- Une méthode d'initialisation de la population des solutions candidates.
- Les valeurs des paramètres de l'algorithme génétique utilisé (par exemple, la taille de la population).
- Les opérateurs génétiques qui produisent l'ensemble des nouveaux individus.
- Le critère d'arrêt de l'algorithme génétique.

La figure 2.7 décrit le squelette d'un algorithme génétique. Chaque itération de l'algorithme correspond à une génération, où une population constituée de plusieurs individus, représentant des solutions potentielles du problème considéré, est capable de se reproduire. Elle est sujette à des variations génétiques et à la pression de l'environnement qui est simulée à l'aide de la fonction d'adaptation (fitness), ce qui provoque la sélection naturelle (la survie du plus fort). Les opérateurs de variation sont appliqués (avec une probabilité donnée) aux individus parents sélectionnés, ce qui génère de nouveaux descendants appelés enfants (ou offsprings) ; on parlera de mutation pour les opérateurs unaires, et de croisement pour les opérateurs binaires (ou n-aires). Les individus issus de ces opérations sont alors insérés dans la population. Le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'une condition d'arrêt soit vérifiée, par exemple, quand un nombre maximum de générations ou un nombre maximum d'évaluations est atteint

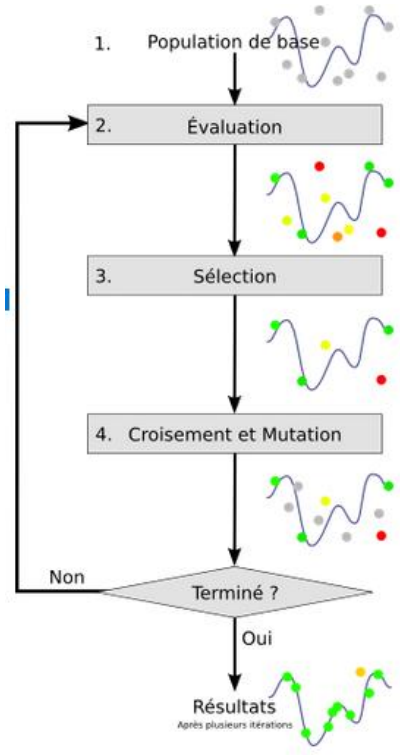


Figure 2.7 : Processus d'un algorithme génétique

4.1 Le codage des données

La représentation des individus est basée sur le codage de l'information, ce dernier concerne les moyens de formaliser l'information afin de pouvoir la manipuler, la stocker ou la transmettre. Il ne s'intéresse pas au contenu mais seulement à la forme et à la taille des informations à coder.

Pour les algorithmes génétiques, le codage est l'un des facteurs les plus importants, c'est la façon dont elles sont codées les solutions. Le codage est la première étape de définir et coder convenablement le problème, cette étape associe à chaque point de l'espace de recherche une structure de données spécifique, appelée génotype (un génotype **:** représente l'ensemble des valeurs des gènes d'un chromosome).

Le codage nécessite une adaptation des opérateurs de croisement et mutation.

Il existe deux types de codage :

Le codage binaire est la représentation la plus fréquente, ce codage a été le premier à être utilisé dans le domaine des AGs. Il présente plusieurs avantages : alphabet minimum {0,1}, facilité de mise au point d'opérateurs génétiques.

L'intérêt principal du codage binaire est de permettre l'utilisation d'opérateurs de croisement et mutation simple.

Le codage réel (appelé aussi codage symbolique) est robuste pour les problèmes considérés comme difficile pour le codage binaire, cela peut-être utile notamment dans le cas où l'on recherche le maximum d'une fonction réelle. Si on veut chercher l'optimum d'une fonction à ***n*** variables

$f(x_1,x_2,\ldots\ldots x_n)$, ou : x_i des entiers, alors l'individu *I* sera de forme :

I :	x_1	x_2	…	x_n

4.2 Génération de la population initiale

L'AG démarre avec une population composée de ***N*** individus dans le codage retenu, le choix des individus conditionne fortement la rapidité de l'algorithme, Il est intéressant que la population soit répartie sur tout l'espace de recherche.

La diversité de la population doit être entretenue aux cours des générations afin d'explorer le plus largement possible l'espace de recherche. C'est le rôle des opérateurs de croisement et de mutation.

4.3 L'évaluation

L'évaluation permet de s'assurer que les individus performants seront conservés, alors que les individus peu adaptés seront progressivement éliminés de la population**.** Elle ne dépendant pas de celle des autres individus.

L'évaluation doit fournir une valeur permettant de déterminer une relation d'ordre entre les différents individus, elle est réalisée par une fonction (la fonction de fitness), qui doit produire objectivement une estimation de la qualité du résultat.

Il est généralement nécessaire de fournir des valeurs de fitness strictement positives pour permettre aux opérateurs de sélection de fonctionner correctement.

4.4 Sélection

Cet opérateur est important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir. Il existe plusieurs techniques de sélection :

Sélection uniforme : On ne s'intéresse pas à la valeur d'adaptation (fitness) et la sélection s'effectue d'une manière aléatoire et uniforme telle que chaque individu ***I*** a la même probabilité ***P(I)*** = 1/***N*** (***N*** la taille de la population).

Sélection par tournoi binaire : Deux individus sont choisis au hasard et on les fait "combattre", celui qui a la fitness la plus élevée l'emporte avec une probabilité ***p*** comprise entre 0.5 et 1.

```
Algorithme Select Tournement
Une population d'individus (Vecteur d'individus ) E (x1, x2,..., xn), n :
est la taille de la population
f (f1, f2,..., fn), vecteur d'évaluation,
t ← tournement, t>1
Best ← tirage avec remise d'un individu de E
Pour i=2,n Faire;
   Next ← tirage avec remise d'un individu de E
   r ← random(0,1)
   Si  f(Next) > f(Best) Alors Best ← Next
Si r> 0.5 Alors retourne Best
FinPour
```

Figure 2.8 : Sélection par tournoi binaire

Sélection par roulette consiste à donner à chaque individu une probabilité d'être sélectionné proportionnelle à sa performance. Un individu avec une forte évaluation a une grande chance d'être sélectionné alors qu'un individu avec un plus faible coût en a moins. Le principe de cette sélection est le suivant :

Si la population d'individus est de taille égale à ***n***, alors la probabilité de sélection d'un individu x_i notée $\boldsymbol{p(x_i)}$ est égale à:

$$P(x_i) = f(x_i) / \sum_{i=1}^{n} f(x_i) \qquad (2.1)$$

En pratique, on calcul pour chaque individu $\boldsymbol{x_i}$ *sa probabilité cumulée* et on choisit aléatoirement un nombre ***r*** compris entre 0 et 1.

$$q_i = \sum_{j=1}^{i} p(x_j) \qquad (2.2)$$

L'individu retenu est x_1 *si* $q_1 \geq r$ *ou* x_i*(* $2 \leq i \leq N$ *)* *si* $q_{i-1} < r \leq q_i$

Ce processus est répété ***N*** fois. Avec une telle sélection, un individu fort peut être choisi plusieurs fois. Par contre, un individu faible aura moins de chance d'être sélectionné.

Algorithme Select Roulette
Une population *d'individus (Vecteur d'individus)* $E(x_1, x_2, \ldots, x_n)$, *n : est la taille de la population*
$f(f_1, f_2, \ldots, f_n)$, *vecteur d'évaluation,*
$P(P_1, P_2, \ldots, P_n)$, *vecteur de* probabilités de sélection des individus
$q_1 \leftarrow P_1$
Pour $i=2,n$ ***Faire***; $q_i \leftarrow q_{i-1} + P_i$
Répéter
  r ← random [0,1]
  Pour $i=2,n$ ***Faire***
    Si $q_{i-1} < r \leq q_i$ **Alors** Retourne x_i
    Sinon Retourne x_1
  FinPour
Jusqu'à *(nombre d'individus sélectionné atteint le Max cherché)*

Figure 2.9 : sélection par roulette

4.5 Croisement

Le croisement utilisé par les algorithmes génétiques est la transposition informatique du mécanisme qui permet, dans la nature, la production de chromosomes qui héritent partiellement des caractéristiques des parents.

L'opérateur de croisement permet la création de nouveaux individus selon un processus fort simple. Il permet donc l'échange d'information entre les chromosomes (individus).

Cet opérateur est appliqué après avoir appliqué l'opérateur de sélection sur la population *P*, on se retrouve donc avec une population P' de *n/2* individus et on doit doubler ce nombre pour que notre nouvelle génération soit complète.

Dans le codage réel l'opérateur de croisement est simple, il suffit de sélectionner un ou plusieurs points de croisement puis fait l'échange de l'information.

Ci-dessous, un schéma illustrant un croisement en un point, un autre pour un croisement en deux points dans le codage binaire.

Croisement à un point

Pour chaque couple, on choisit au hasard un point de croisement (figure 2.5). Le croisement s'effectue directement au niveau binaire, et non au niveau des gènes. Un croisement peut être coupé au milieu d'un gène.

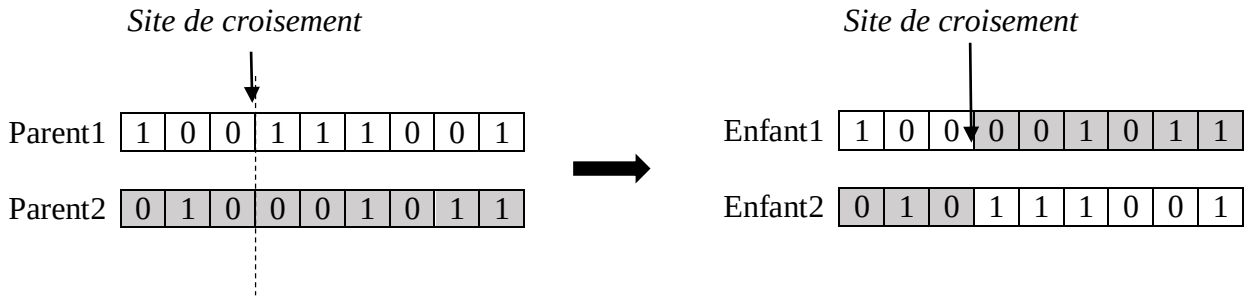


Figure 2.10 : Représentation schématique du croisement en un point

Croisement à un deux points

On choisit au hasard deux points de croisements successifs. Cet opérateur est généralement considéré comme plus efficace que le précédent (croisement à un point)

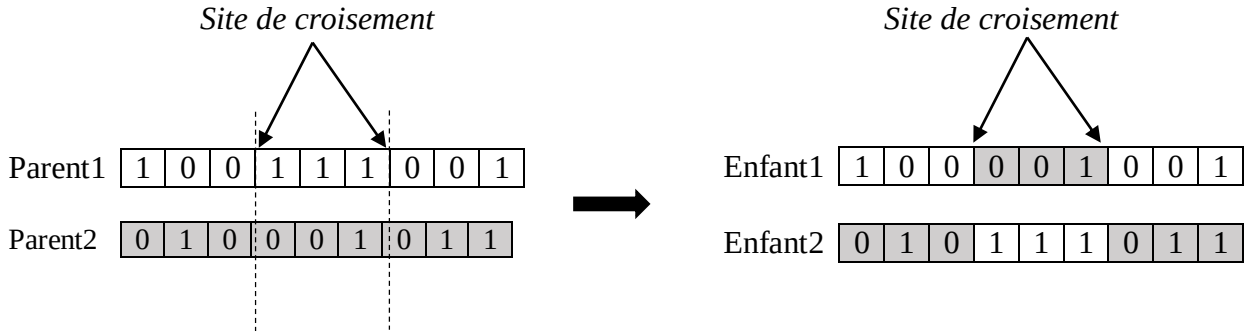


Figure 2.11 : Représentation schématique du croisement en deux point

Le codage réel requiert des opérateurs génétiques spécifiques pour la manipulation des chromosomes. Il est de plusieurs types : l'opérateur de Croisement PMX (Partially Mapped Crossover), L'opérateur de Croisement CX (Cycle Crossover)…

L'opérateur de croisement Partially Mapped Crossover (PMX)

PMX est un opérateur de croisement à **deux points de coupure** qui définit un segment de même longueur dans chacun des parents P1 et P2.

Un exemple de l'opérateur de Croisement PMX est illustré à la figure ci-dessous. Les segments de parents (a) ont copiés vers les enfants E1 et E2. E1 a hérité du segment de P2 et E2 de P1(b). A

partir de l'un de ces segments, on établit à chaque gène une liste de correspondance. Cette liste va servir à placer les gènes redondants et elle est formée de la manière suivante : pour chaque position du segment on note x le gène qui s'y trouve et y celui de l'autre enfant dans la même position. Tant que y est retrouvé ailleurs dans le segment de départ, on note y' son correspondant dans l'autre enfant et on remplace y par y'[6].

On procède ensuite au placement des gènes hors segment en les copiant des parents respectifs. Si le placement d'un gène provoque un conflit puisque ce gène existe déjà. On utilise alors son correspondant dans la liste. En procédant de manière itérative, on arrive à former les enfants E1 et E2 illustrés dans (c) de la figure suivante[10] :

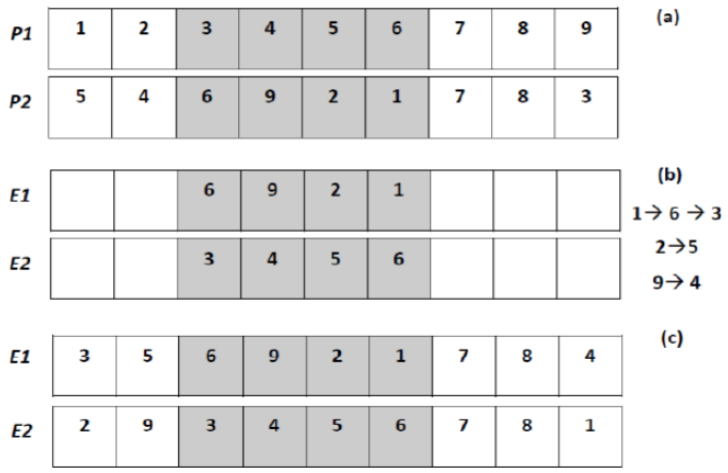


Figure 2.12 : Représentation schématique du croisement PMX

L'opérateur de croisement Cycle Crossover (CX)

CX est un opérateur qui satisfait la condition suivante : chaque gène d'un enfant provient de l'un des parents à la même position. Les enfants sont donc formés en copiant un gène d'un parent et en éliminant l'autre à la même position puisqu'il va appartenir au deuxième enfant. Une fois que les positions occupées sont copiées par élimination, on a complété un cycle. Les places restantes des deux enfants sont complétées par les parents opposés [6]. (voir l'exemple suivant) :

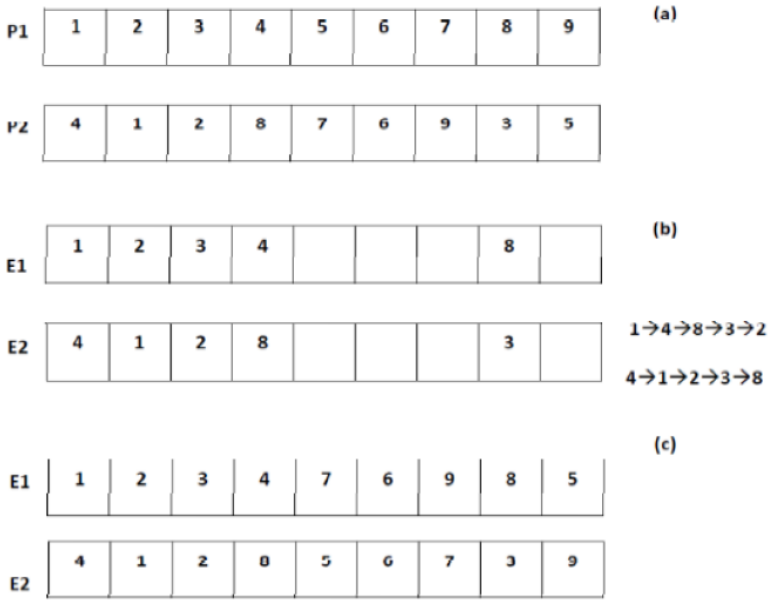


Figure 2.13 : Représentation schématique du croisement CX

4.6 Mutation

La mutation est un changement aléatoire selon une certaine règle probabiliste qui doit faire sur les génotypes, avec une faible probabilité P_m (fixée par l'utilisateur), La probabilité P_m de mutation selon Fonseca est calculée comme suivant :

$$P_m = 1 - \sigma^{-1/l} \quad (2.3)$$

l : est le longueur du chromosome.
σ : est la pression sélective, où sa valeur recommandée est où sa valeur recommandée est: 1,8.

La mutation classique consiste à transformer dans un chromosome binaire un 1 en un 0 ou le contraire.

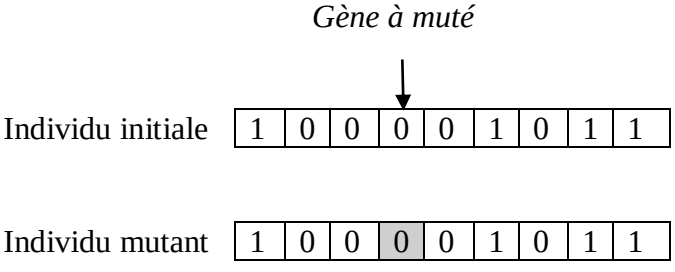


Figure 2.14 : Représentation schématique de mutation

Pour le codage réel, les opérateurs de mutation les plus utilisés sont les suivants :

Mutation par inversion

Deux positions sont sélectionnées au hasard et tous les gènes situés entre ces positions sont inversés.

L'individu ***I*** avant mutation est représenté dans la partie (**a**) avec le segment formé par les deux positions sélectionnées ***P1***et ***P2***. L'inversion est effectuée à l'étape (**b**) afin de produire l'individu ***I'.***

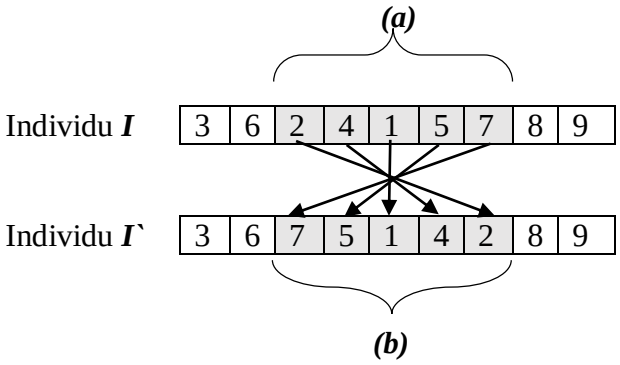


Figure 2.15 : Représentation schématique de la mutation par inversion

Mutation par déplacement

Une séquence est sélectionnée au hasard et déplacée vers une position elle-même tirée au hasard. Un exemple de mutation par déplacement est illustré à la figure suivante, déplacement de la partie (a) dans la première position.

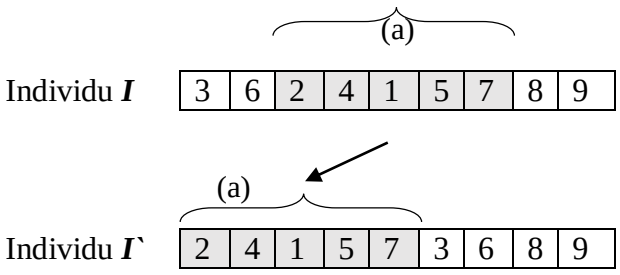


Figure 2.16 : Représentation schématique de la mutation par déplacement

Mutation par permutation

Deux positions ***P1*** et ***P2***sont sélectionnées au hasard et les gènes situés dans ces positions sont permutés. Comme il est illustré à la Figure suivante

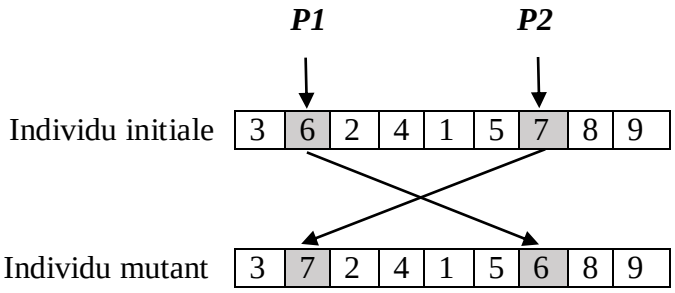


Figure 2.17 : Représentation schématique de la mutation par permutation

4.7 Remplacement

Cette étape sert à remplacer la population de l'instant ***t*** (itération ***t***) par la nouvelle population de l'instant ***t+1***. C'est la dernière étape de chaque itération jusqu'au critère d'arrêt suffisant.

Une stratégie élitiste consiste à conserver au moins un individu de la population de la génération précédente, dans la génération suivante

5. Conclusion

Dans ce chapitre, nous avons présenté les problèmes d'optimisation et les différentes approches de leur résolution. Nous avons insisté beaucoup plus sur la méta-heuristique algorithme génétique que nous allons l'adopter pour résoudre le problème de fragmentation d'ADN qui sera montré dans le chapitre suivant.

CHAPITRE 3 :

Conception

1. Introduction

Le problème d'assemblage de fragments (FAP) consiste à construire une séquence d'ADN à partir de plusieurs centaines (voire des milliers) de fragments obtenus par les biologistes du laboratoire. Il s'agit donc d'un problème d'optimisation combinatoire, connu comme étant NP-difficile ; pour *n* fragments, il y a $2^n n!$ configurations possibles, ou 2^n est le nombre de toutes les combinaisons possibles en termes d'orientation de fragments, et $n!$ est le nombre de toutes les permutations possibles des fragments.

Dans ce projet, nous adoptons l'algorithme génétique (AG) pour résoudre le problème d'assemblage de fragments. Nous tentons d'améliorer la recherche de bonne solution par l'application de l'algorithme PALS (Alba et Luque, 2007)[3] comme un opérateur de recherche additionnel dans l'AG ; après que l'algorithme AG localise une bonne région de solutions dans l'espace de recherche (exploration), PALS permet l'amélioration (exploitation) de cette région via la recherche locale.

Dans ce chapitre, nous décrivons la conception de notre système de résolution du problème d'assemblage de fragments d'ADN par l'hybridation de l'algorithme génétique avec PALS.

2. Hybridation de l'AG avec PALS

L'algorithme PALS (pour Problem Aware Local Search) a été proposé par Alba et Luque en 2007 (Alba et Luque, 2007)[3]. C'est une méthode de recherche locale qui fait évoluer une seule solution en utilisant la notion de voisinage et des critères permettant de définir le sous-ensemble de solutions acceptables parmi toutes les solutions voisines d'une solution courante.

Dans un algorithme hybride AGPALS, l'algorithme PALS est appliqué comme un opérateur de recherche additionnel dans un AG. Dans cet algorithme AGPALS les nouvelles solutions produites par l'operateur de croissement et l'operateur de mutation, sont exploitées pour fournir des configurations initiales pour PALS. La figure suivante montre l'architecture globale de cette hybridation.

Dans notre projet, nous avons travaillé avec des instances de fragments d'ADN générés artificiellement par l'outil GenFrag [2] et les informations de séquences choisies sont disponibles sur le site du NCBI [22].

L'objectif est de chercher un ordre total des fragments donnés qui aboutit à une séquence consensus reflétant avec précision la séquence parent. Si la séquence parent est préalablement

connue, on peut juger la qualité d'un résultat d'assemblage par une mesure de ressemblance entre la séquence consensus finale et la séquence parent connue.

3. Description détaillée de l'AGPALS

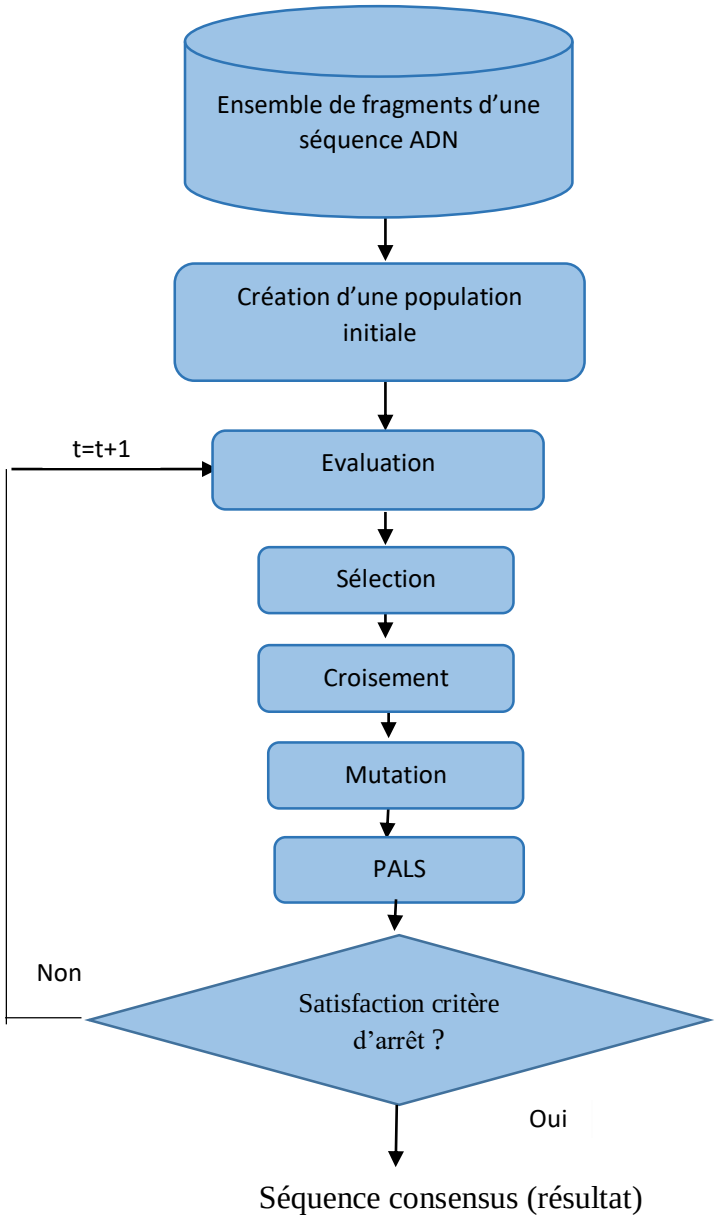


Figure 3.1 : digramme de l'algorithme AGPALS

3.1. Génération de la population initiale

Soient un ensemble de fragments d'ADN numérotés de 1 jusqu'à p, nous voulons trouver un ordre total des fragments donnés qui aboutit à une séquence consensus reflétant avec précision la séquence parent. Si la séquence parent est préalablement connue,

La solution de ce problème peut être représentée par un vecteur (de dimension p) des numéros de fragments chevauchants qui doivent mettre l'un après l'autre pour trouver la séquence complète de génome. Par exemple, si nous avons un ensemble *F* de 10 fragments (*p*=10) numérotés de 0 jusqu'à 9 ; *F*= {0,1,2,3,4,5,6,7,8}. Une solution, de ce problème, est une permutation des éléments de cet ensemble. Une population de solutions peut être un ensemble de combinaisons possibles des éléments cet ensemble.

Exemple : une population initiale de taille 6, avec les numéros de fragments sont de 0 à 9.

Solution1	1	5	0	3	2	6	4	7	8
Solution2	8	6	1	5	7	0	3	4	2
Solution3	8	6	1	5	7	0	3	4	2
Solution4	5	7	1	3	8	4	2	0	6
Solution5	1	2	3	4	5	6	7	8	9
Solution6	8	9	5	7	2	1	3	4	0

Figure 3.2 : schéma illustratif d'une population initiale

L'AG démarre avec une population initiale de ***N*** individus (solutions) générés de façon aléatoire. La taille de la population *N* est donnée par l'utilisateur.

3.2. Evaluation de la population

Chaque individu de la population est évalué par une fonction objectif (fitness) qui représente sa performance. Dans notre projet, la fonction objectif (fitness) représente la somme de chevauchements de chaque deux fragments successifs dans la solution. Le chevauchement entre

deux fragments de la solution *S*, est le nombre de nucléotides commun de préfix de fragment *S* (***i***) et suffixe de fragment *S* (***i+1***). La fonction objectif de la solution *S* est notée *f* :

$$f(S) = \sum_{i=0}^{n-1} W(S(i), S(i+1)) \qquad (3.1)$$

- ***S*** : une solution (individu) à l'instant ***t***

- **n** : nombre de fragments

- **S*(i), S(i+1)*** : deux fragments ***successif***

- $W(S(i), S(i+1))$: chevauchement entre les fragments S(***i***) et S(***i+1***).

Exemple illustratif

Supposant, dans un instant ***t,*** on a la population de trois solutions et le nombre de fragments égale à 9) :

Solution1(S1)	1	5	0	3	2	6	4	7	8
Solution2(S2)	8	6	1	5	7	0	3	4	2
Solution3(S3)	2	4	1	5	6	0	7	3	8

Avec les fragments suivants :

F0 : GTCAATT
F1 : CTAGGAACTA
F2 : TGCG
F3 : TTACATTGCG
F4 : CCAATTGA
F5 : AACTATGT
F6 : CGTAATTGCC
F7 : GATACCG
F8 : CGTAATT

On calcule la fitness pour chaque solution.

Solution1 : (S1)

1	5	0	3	2	6	4	7	8

Le schéma suivant, présenté l'alignement des 9 fragments de la solution1 :

F1 : CTAGGAACTA
F5 : AACTATGT
F0 : GTCAATT
F3 : TTACATTGCG
F2 : TGCG
F6 : CGTAATTGCC
F4 : CCAATTGA
F7 : GATACCG
F8 : CGTAATT

$f(S1)$= chevauchement(1,5)+ chevauchement(5,0)+ chevauchement (0,3)+ chevauchement(3,2)+ chevauchement(2,6)+ chevauchement(6,4)+ chevauchement(4,7)+ chevauchement(7,8)

=1+2+2+4+2+2+2+3=**18**

Solution2: (S2)

8	6	1	5	4	7	0	3	2

F8 : CGTAATT
F6 : CGTAATTGCC
F1 : CTAGGAACTA
F5 : AACTATGT----
F4 : CCAATTGA
F7 : GATACCGT
F0 : GTCAATT
F3 : TTACATTGCG----
F2 : TGCG

$f(S2)$= = chevauchement(8,6)+chevauchement(6,1)+ chevauchement(1,5)+ chevauchement(5,7)+ chevauchement(7,0)+ chevauchement(0,3)+ chevauchement(3,4)+ chevauchement(4,2)
= 7+1+5+0+2+2+2+0=**19**

3.3. Sélection

Cet opérateur est peut-être le plus important puisqu'il permet aux individus d'une population de survivre, de se reproduire ou de mourir.

Dans notre projet, nous avons choisi la sélection par tournoi, son principe est le suivant :

Pour sélectionner un individu, on en tire *t* (*t* la taille du tournoi) uniformément dans la population, et on sélectionne le meilleur de ces *t* individus.

3.4. Croisement

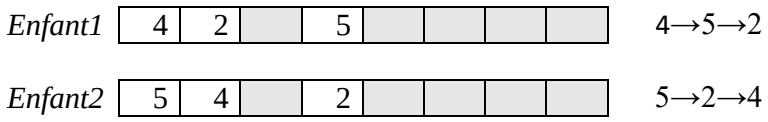
Dans notre projet, nous avons appliquée l'opérateur de croisement *CX* (Cycle Crossover), pour le codage réel, dont le principe est illustré dans l'exemple suivant :

Exemple :
On a deux solutions parentes (solutions) ***parent1*** et ***parent2*** :

Parent1	4	2	6	5	1	3	7	8

Parent2	5	4	1	2	3	6	8	7

Après l'application d'opérateur CX :



Alors les enfants sont :

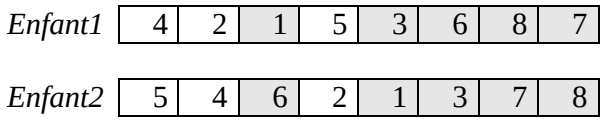


Figure 3.3 : schéma illustratif de croisement CX

3.5. Mutation

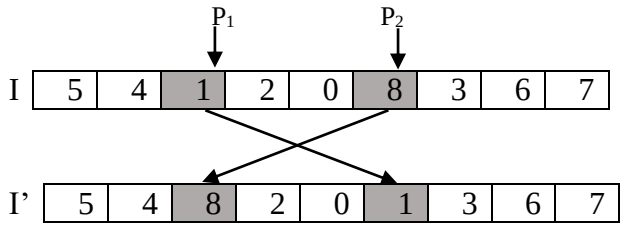
Nous avons appliqué l'operateur de mutation par permutation à chaque enfant.

Le principe de la mutation par permutation est :

- Générer deux nombres aléatoires pt_1 et pt_2 ; $pt_1, pt_2 \in (1, n)$, ***n*** est le nombre de fragments.
- Faire une permutation entre les gènes pt_1 et pt_2

Exemple :

On a l'individu ***I***, et deux points (pt_1, pt_2) générés de façon aléatoire , par exemple pt_1=3 et pt_2=***6***



I' est l'individu ***I*** après l'opération de mutation.

Figure 3.4 : schéma illustratif de la mutation par permutation

3.6. Algorithme PALS

L'algorithme PALS permet d'améliorer chaque solution enfant *S* produite par l'operateur de mutation, on la remplace par une bonne solution ***S`*** appartenant à son voisinage *N(S).* son le principe est le suivant :

- Générer un ensemble *N(s)* de solutions voisines de la solution *S*, par l'application des mouvements sur la solution courante *S*. Un mouvement consiste à inverser l'ordre des fragments situés entre deux positions distinctes ***i*** et ***j*** dans solution ***S***.
- Sélectionner la meilleure solution ***S`*** appartenant à l'ensemble de voisinage *N(S)*. Nous utilisons le nombre de contigs comme un premier critère pour ordonner les mouvements et pour juger la qualité des solutions. Dans le cas d'égalité, il considère le niveau de chevauchement maximal entre les fragments successifs pour parfaire l'évaluation. Dans ce cas, nous sélectionnons la meilleure solution voisine *S'* [16] qui réalise le critère suivant :

$$nContigs(S) < nContigs(S') \text{ ou } (nContigs(S) = nContigs(S') \text{ et } f(S) > f(S')) \qquad (3.2)$$

Tels que :

nContigs : c'est une fonction qui donne le nombre de contigs dans une solution.

f: c'est une fonction qui calcule le nombre de chevauchement entre les fragments d'une solution selon l'équation (3.1).

Exemple illustratif

Soit *S2* une solution voisine de la solution *S1*

Solution1

1 5 0 3 2 6 4 7 8

Fitness=18

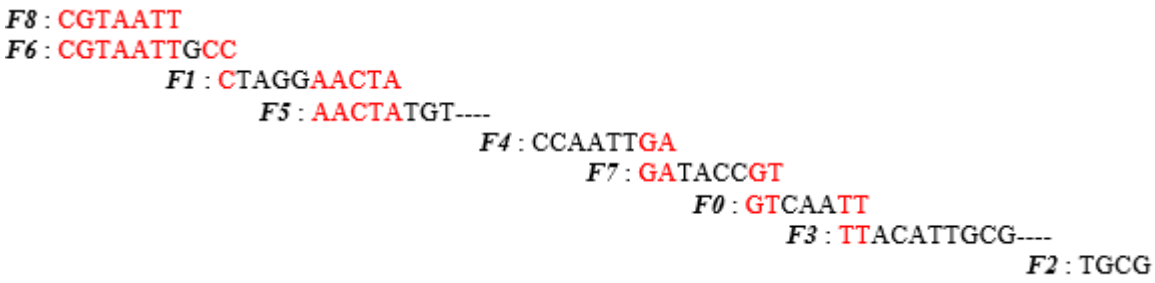
Nombre de contig = 1 car tous les fragments sont chevauchés

F1 : CTAGGAACTA
F5 : AACTATGT
F0 : GTCAATT
F3 : TTACATTGCG
F2 : TGCG
F6 : CGTAATTGCC
F4 : CCAATTGA
F7 : GATACCG
F8 : CGTAATT

Solution2

8 6 1 5 4 7 0 3 2

Fitness= 19



Nombre de contig = 3 car pas de chevauchement entre tous les fragments.

Solution	*f*	*nContigs*
1	18	1
2	19	3

Tableau 3.1 : exemple illustratif des résultats de PALS

Selon l'algorithme PALS :

- Si nombre de contigs de solution ***S1*** et inférieur de celui de ***S2*** alors ***S*** est la meilleure.
- Si le nombre de contigs est le même alors la meilleure solution est celle qui a la meilleure fitness *f*.

ET dans notre exemple :

Nombre de contigs (***solution1***) < Nombre de contigs (***solution2***)

Alors : La meilleure solution est ***solution1***

3.7 Critère d'arrêt

L'algorithme AGPALS est itératif, à chaque itération, les solutions obtenues par application successive des opérateurs de : sélection, croisement, mutation et PALS vont remplacer l'ancienne la population (population de parents). Le critère d'arrêt est considéré comme le nombre d'itération qui est donné par l'utilisateur.

4. Conclusion

Dans ce chapitre, nous avons présenté le thème de notre projet. Nous avons décrit notre résolution au problème d'assemblage de fragments d'ADN à l'aide des algorithmes et des exemples.

Le chapitre suivant sera consacré a l'implémentation de ces algorithme et la description des résultats obtenus.

CHAPITRE 4 :

Implémentation

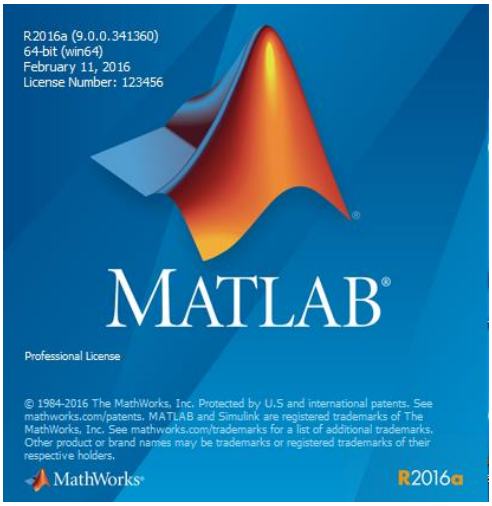
1. Introduction

L'implémentation de n'importe quel programme ou application nécessite un ensemble d'étape, qui joue un rôle très important à sa performance. L'une des étapes la plus importante l'environnement de travail, le choix de dernier doit être par conscience, selon les structures nécessaires à l'implémentation. Ainsi que les outils de travail qui facilitent la tâche de programmation et d'utilisation des données.

2. Environnement et outils de développement

2.1. Environnement de développement

Pour l'implémentation de notre assembleur de fragment d'ADN, nous avons choisi l'environnement Matlab,



Matlab est un logiciel de calcul numérique commercialisé par la société MathWorks1. Il a été initialement développé à la fin des années 70 par Cleve Moler, professeur de mathématique à l'université du Nouveau-Mexique puis à Stanford, pour permettre aux étudiants de travailler à partir d'un outil de programmation de haut niveau et sans apprendre le Fortran ou le C.

MATLAB est une abréviation de *Matrix LABoratory*. C'est un langage pour le calcul scientifique, l'analyse de données, leur visualisation et le développement d'algorithmes.

Matlab trouve ses applications dans de nombreuses disciplines. Il constitue un outil numérique puissant pour la modélisation de systèmes physiques, la simulation de modèles mathématiques, la conception et la validation (tests en simulation et expérimentation) d'applications. Le logiciel de base peut être complété par de multiples *toolboxes*, c'est-à-dire

des boîtes à outils. Celles-ci sont des bibliothèques de fonctions dédiées à des domaines particuliers. Nous pouvons citer par exemple : l'Automatique, le traitement du signal, l'analyse statistique, l'optimisation…

2.2. Outils utilisés

Nous avons utilisé les instance d'ADN fournis par l'outil GenFrag , il est conçu comme mesure de comparaison des assembleurs (algorithme d'assemblage) en termes d'efficacité et de qualité de solution, il génère artificiellement des instances d'ADN. Ces benchmarks sont issus d'un travail de L.Michael Engle et Christian Burk[2].

Les fragments générés possèdent un seuil minimal de chevauchement égal à 30pb, ce qui rend l'assemblage un peu difficile pour les instances de grande taille.

Les instances de GenFrag

Instance	Moyenne de longueur des fragments	Nombre de fragments	Couverture	La langueur de la séquence
X60189-4	395	39	4	3835
X60189-5	286	48	5	
X60189-6	343	66	6	
X60189-7	387	68	7	
m15421-5	398	127	5	10089
m15421-6	350	173	6	
m15421-7	383	177	7	
j02459-7	405	352	7	77292
38524243_4	708	442	4	
38524243_7	703	773	7	

Tableau 4.1 : Tableau des instances de GenFrag

GenFrag est un ensemble d'instances d'ADN de taille différente ; chaque instance contient un nombre spécifique de fragments. Elle se trouve dans un répertoire qui prend le nom de l'instance, par exemple :

- Nom d'instance: x60189_4, x60189_5, x60189_6, x60189_7 m15421_5, m15421_6, m15421_7, j02459_7, 38524243_4 (bx842596_4), 38524243_7 (bx842596_7).

Le répertoire représentant l'instance contient un ensemble d'informations décrivant l'instance. Par exemple :

- La séquence cible d'ADN et des informations (nombre de nucléotides qui la forme,…..)

- X.dat - fichier de fragments dans un format similaire à ***fasta*** (un fichier ***fasta*** et formé d'une entête et une séquence de nucléotides).

- frag_X.dat - fichier de fragments correspondant au génome de l'ADN utilisé pour générer le benchamark (numéro de début et de fin de la série des nucléotides forment le fragment).

- un fichier *exel* présente les chevauchement entre les fragments (l'intersection de la ligne ***i*** et la colonne ***j*** présent chevauchement entre les fragments ***i*** et ***j*** .
- D'autre information concernant la matrice de chevauchement.

2.3. Les structures de données utilisées

Pour la manipulation des données, nous avons choisi une structure sous une forme multidimensionnelle qui convient mieux au langage de programmation et la nature des données utilisées.

L'individu *I* est représenté par un tableau d'entier d'une seule dimension.

I :	x_1	x_2	…	x_n

Ou : x_i les numéros des fragments de la séquence.

La population des individus est représentée par un tableau multidimensionnel de déférents types de champs :

- Le champ *numéro* contient le numéro de l'individu.
- Le champ *fitness* contient la fitness de l'individu.
- Le champ *individu* contient l'individu (solution).

L'accès aux éléments de cette structure s'effectue par l'utilisation de nom de tableau et nom de chaps (*NomTableau.nomChamp)*.

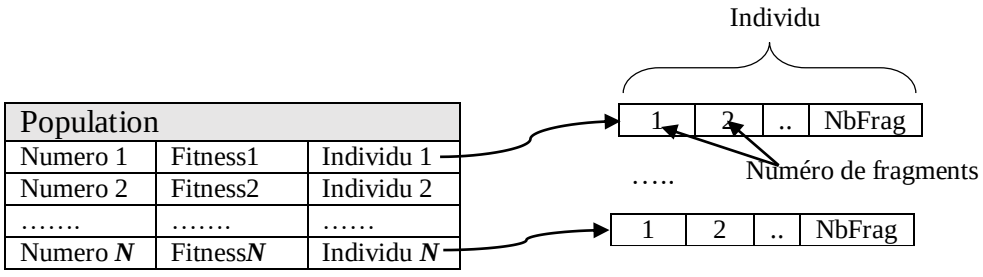


Figure 4.1 : Structure d'une population

N : nombre de population
NbFrag : nombre de fragments de l'instance d'ADN

Les chevauchements entre les fragments sont représentés par Une matrice carrée de taille $n \times n$ (n est le nombre de fragments).

Les fragments utilisés sont représentés dans un tableau multidimensionnel contient deux champs :

- Un champ pour le numéro de fragment.
- Un champ contient le fragment lui-même (un fragment c'est un texte de quatre lettre A, C, T et G).

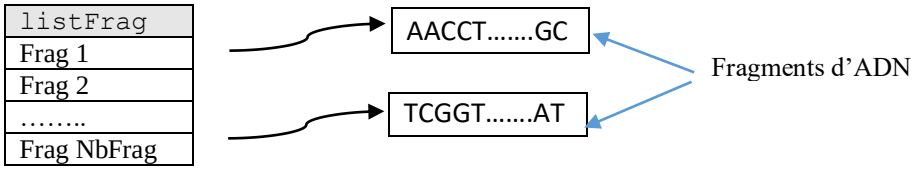


Figure 4.2 : Structure de la liste de fragments

2.4 Les algorithmes

La réalisation de ce travail est faite à l'aide de l'hybridation de l'algorithme génétique (AG) et l'algorithme PALS.

2.4.1 Algorithme génétique (AG) :

L'implémentation de cet algorithme nécessite un ensemble d'étapes.

Algorithme 4.1 : Algorithme génétique (N, t)

N : taille de la population initiale
t : nombre d'itérations

Début

Génération_Population_initiale (N, instTest)
Calcul_fitness (N,NbFrga, chevauch)
Algorithme_sélection (population,N, r,f)
Croisement CX (liste_parents, NbFrag)
Mutation (NbFrag)
Remplacement

Fin

2.4.2 L'algorithme PALS

L'utilisation de cet algorithme permet d'améliorer chaque solution enfant *S* produite par l'operateur de mutation, on la remplace par une bonne solution ***S`*** appartenant à son voisinage *N(S)*

Algorithme 4.2 : Algorithme PALS (*S, NbFrag*)

S : une solution
NbFrag : nombre de fragments

Debut
Pour i=1 ← NbFrag-1 ***faire***
Pour j ← i+1 Nbfrag ***faire***
calcul_deltaF_deltaC (S, chevauch, cutoff, NbFrag)
fin pour
fin pour
retourne liste_mouvement *{la liste des mouvements de la solution}*
select_mouvement (liste_mouvement, chevauch, cutoff,S)
Fin

Algorithme PALS, se base sur une le calcul de *deltaF* et *deltaC.*

deltaF est la variation de la longueur totale de chevauchement, il est calculé par la soustraction de la longueur de chevauchement des fragments affectés de la solution courante, de celle de la solution modifiée

deltaC est la variation du nombre de contigs, la valeur de deltaC est calculée en testant, après l'application d'un mouvement, si un contig courant est coupé ou non, en se basant sur la valeur d'un paramètre *cutoff*, qui représente la longueur de chevauchement minimale pour considérer que deux fragments adjacents étant dans le même contig .

Algorithme 4.3 : Calcul_deltaF_deltaC (S, chevauch, cutoff, NbFrag)

S : solution
Chevauch : matrice des chevauchements
NbFrga : nombre de fragments
Cutoff : seuil de chevauchement
Debut
Calculer deltaF
Calculer deltaC
Mémoriser mouvement *{la liste des mouvements}*
Retourne liste_mouvement
Fin

2.4.3 L'algorithme AGPALS (algorithme génétique (AG) hybridé avec l'algorithme PALS).

L'hybridation de deux algorithmes algorithme génétique (GA) et le *problème Aware recherche locale* (PALS**)** est présentée dans l'algorithme suivant :

Algorithme 4.4 : Algorithme AGPALS (***N, t***)

N : taille de la population
t : nombre d'itérations

Début

Génération_Population_initiale (N, instTest)
Calcul_fitness (N,NbFrga, chevauch)
Algorithme_sélection (population,N, r,f)
Croisement CX (liste_parents, NbFrag)
Mutation (NbFrag)
PALS(S, chevauch,cutoff)
Remplacement

Fin

3- Présentation détaillée de de l'algorithme AGPALS.

3.1 Génération d'une population initiale

La première étape de cet algorithme est générer aléatoirement une population initiale.

Algorithme 4.5 : Génération_Population_initiale (N, instTest)

N : taille de la population
instTest : fichier de fragments

Debut

- Choix de l'instance.
- Lecture des fragments
- Lire le nombre de population (*N*)
- Extraire nombre de fragments (NbFrag)
- Générer (*N*) individus aléatoires.

Fin

Retourne ***P*** *{P : population initiale}*

La description de cet algorithme est la suivante :

Choix de l'instance

Le choix de l'instance nécessite l'accès et l'ouverture de fichier contenant ces fragments, à l'aide de la fonction matlab '`fopen`':

```
instTest=fopen('chemin\nomInstance.dat','r')
```

Cette fonction retourne une valeur égal ou supérieur à 3 si le fichier existe, et -1 dans le cas contraire.

Lecture des fragments

Après l'accès à l'emplacement du fichier contenant les fragments, la lecture se faite ligne par ligne (chaque ligne représente un fragment), puis le fragment lu sera stocké dans une structure nommée '`listFrag`' présentée dans figure 4.2. La lecture se fait selon l'algorithme suivant :

Algorithme 4.6 : Lecture_fragments (***instTest***)

instTest : fichier des fragments

Début

Ouvrir fichier de fragments
i ← 2; j ← 1 ; *{i : l'indice des lignes}*
Tant que pas fin de fichier ***faire***
listFrag(j) ← ligne (i)
i ← i+2 ; *{sauter la ligne contient le numéro de fragment}*
j ← j+1 ;
Fin
Fermer ***instTest***
Retourne ***NbFrag*** {*NbFrag* : nombre de fragments}
Retourne ***listFrag*** {*ListFrag* : liste de fragments}

Fin

Dans ***matlab*** le processus de lecture des fragments est le suivant :

```
j=1
tant ~feof(instTest)
    listFrag(j).frag= fscanf(instTest, '%s' ,i);
    i=i+2;j=j+1;
end
```

Générer (*N*) individus aléatoires.

C'est l'étape de la création de la population initiale, elle nécessite comme paramètres, la taille de population(***N***) et le nombre de fragments de la séquence (*NbFrag*) qui est déterminé à partir de la lecture de fragments. Les individus générés sont stockés dans une structure nommée '*population'* présentée dans la figure 4.1., le processus qui génère les individus de la population est montrée par l'algorithme suivant :

Algorithme 4.7 : Générer_individus_aléatoires (*N, NbFrag*)

N : taille de la population
NbFrag : nombre de fragment de l'instance

Début

Lire ***N***
Pour i=1,***N*** ***faire***
Générer individu aléatoire ***I*** {individu de taille *NbFrag*}
P(i) ← individu ***I***
Fin pour

Retourne ***P*** *{population initiale}*

Fin

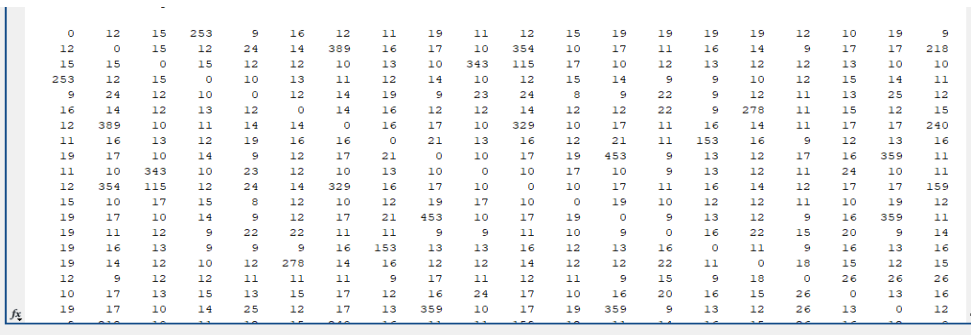
La commande matlab permant de générer aléatoirement un individu de taille 'NbFrag' (NbFrag : le nombre de fragments) est la suivante :

```
individu = randperm(NbFrag)
```

Calcul de fitness

La fitness se calcule à partir de la matrice des chevauchements nommée *'chevauch'* .

Dans la figure 4.3, (une partie de la matrice (442×442) des chevauchements de l'instance '38524243_4') est présentée. Cette dernière est une transformation d'un fichier avec extension '*.exl*' à un fichier avec extension *'.csv'*.



0	12	15	253	9	16	12	11	19	11	12	15	19	19	19	19	12	10	19	9
12	0	15	12	24	14	389	16	17	10	354	10	17	11	16	14	9	17	17	218
15	15	0	15	12	12	10	13	10	343	115	17	10	12	13	12	12	13	10	10
253	12	15	0	10	13	11	12	14	10	12	15	14	9	9	10	12	15	14	11
9	24	12	10	0	12	14	19	9	23	24	8	9	22	9	12	11	13	25	12
16	14	12	13	12	0	14	16	12	12	14	12	12	22	9	278	11	15	12	15
12	389	10	11	14	14	0	16	17	10	329	10	17	11	16	14	11	17	17	240
11	16	13	12	19	16	16	0	21	13	16	12	21	11	153	16	9	12	13	16
19	17	10	14	9	12	17	21	0	10	17	19	453	9	13	12	17	16	359	11
11	10	343	10	23	12	10	13	10	0	10	17	10	9	13	12	11	24	10	11
12	354	115	12	24	14	329	16	17	10	0	10	17	11	16	14	12	17	17	159
15	10	17	15	8	12	10	12	19	17	10	0	19	10	12	12	11	10	19	12
19	17	10	14	9	12	17	21	453	10	17	19	0	9	13	12	9	16	359	11
19	11	12	9	22	22	11	11	9	9	11	10	9	0	16	22	15	20	9	14
19	16	13	9	9	9	16	153	13	13	16	12	13	16	0	11	9	16	13	16
19	14	12	10	12	278	14	16	12	12	14	12	12	22	11	0	18	15	12	15
12	9	12	12	11	11	11	9	17	11	12	11	9	15	9	18	0	26	26	26
10	17	13	15	13	15	17	12	16	24	17	10	16	20	16	15	26	0	13	16
19	17	10	14	25	12	17	13	359	10	17	19	359	9	13	12	26	13	0	12

Figure 4.3 : Matrice des chevauchements ('chevauch')

Un fichier *.csv* est fichier de données numériques séparées par des virgules ‘,’ avec ou sans entête.

Le but de la création de la matrice de chevauchement est de minimiser l’accès fréquent au fichier des chevauchements pour la lecture. La commande matlab ‘`csvread`’ réalise cette transformation :

```
chevauch=csvread('chemin\nom_fihier.csv',2,0);
```

Chaque élément W_{ij} de la matrice représente le chevauchement entre les fragments ***i*** et ***j*** dans cet ordre.

Ramarque : $W_{ij} \neq W_{ji}$

Le calcul de la fitness est fait selon l’algorithme suivant

Algorithme 4.8 : Calcul_fitness (S,NbFrag, chevauch)

S : solution
NbFrag : nombre de fragments
Chevauch : matrice de chevauchements

Début

fitness ← 0 ; { fitness de la solution}
Pour i=1, NbFrag – 1 ***faire***

fitness ← fitness + chevauch(S(i),S(i+1))

Fin pour

Retourne fitness *{fitness de la solution }*

Fin

Exemple illustratif

La figure suivante présente une population initiale avec une taille N=8 et nombre de fragment NbFrag=5.

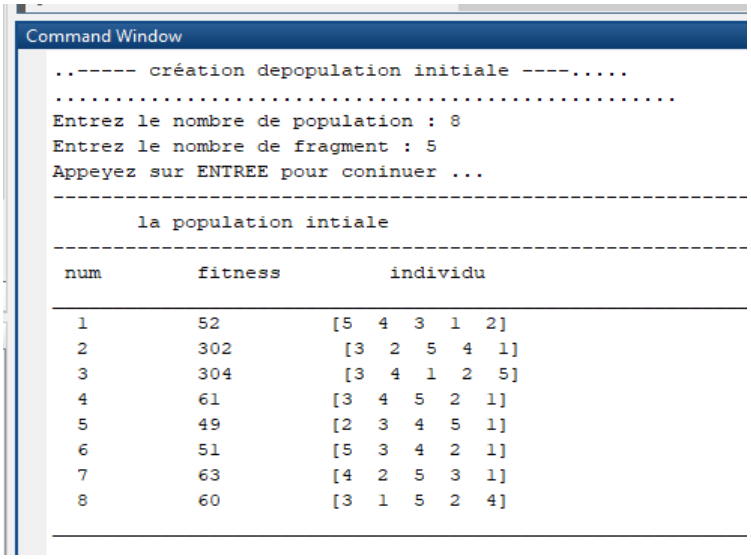


Figure 4.4 : Exemple illustratif d'une population initiale

3.2 Sélection

La sélection est réalisée selon l'algorithme de sélection par tournoi binaire présenté dans la figure 2.1

Pour choisir les parents de reproduction.

Exemple illustratif : (on garde la même population présentée dans l'exemple précédant),).

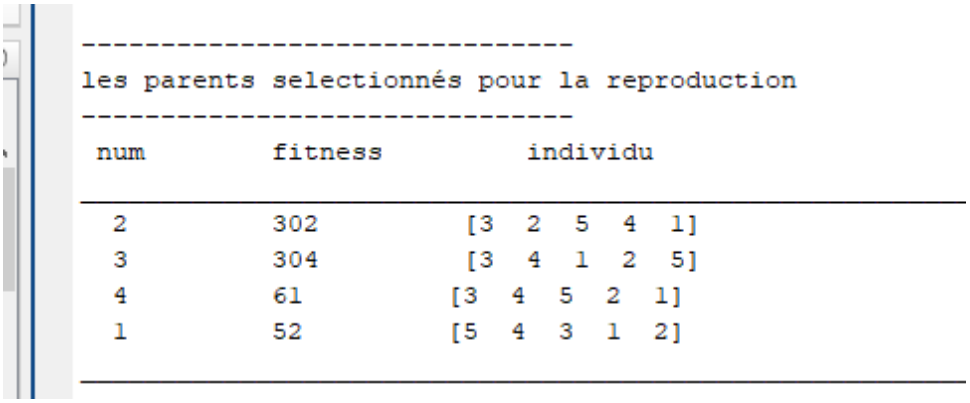


Figure 4.5 : Exemple illustratif de résultat de sélection

3.3 Croisement

L'opérateur de croisement est réalisé selon le principe CX (Cycle Crossover), comme il est montré dans l'algorithme suivant :

Algorithme 4.9 : Croisement CX (liste_parents, NbFrag)

Liste_parents : liste des parents résultante de la sélection
NbFrga : nombre de fragments

Début

enfant1(1) ← Parent1(1)
enfant2(1) ← parent2(1)
contenu ← parent2(1)
i ← 1
Tant que enfant1(1) ≠ contenu ***faire***
i ← i+1
Si parent1(i) = contenu ***alors***
enfant1(i) ← parent1(i)
enfant2(i) ← parent2(i)
contenu ← parent2(i)
Fin si
i ← 1
Fin TQ

i ← 1

Tant que enfant1(i) vide ou i <= *NbFrag* ***faire***
enfant1(i) ← parent2(i)
enfant2(i) ← parent1(i)
i ← i+1
Fin Tant que
Retourne enfant1 *{enfant1 : une solution}*
Retourne enfant2 *{enfant2 : une solution}*

Fin

Exemple illustratif : un exemple d'application d'opérateur de croisement CX

```
parent1=  4  2  6  5  1  3  7  8
parent2=  5  4  1  2  3  6  8  7

enfant1=  4  2  1  5  3  6  8  7
enfant2=  5  4  6  2  1  3  7  8
>>
```

Figure 4.6 : Exemple illustratif de résultat de croisement CX

3.4 Mutation :

Nous avant appliqués la mutation par permutation, qui se base sur l'algorithme suivant :

Algorithme 4.10 : Mutation (S,NbFrag)

S : solution
NbFrag : nombre de fragments

Début

Générer aléatoirement *P1* <= Nbfrag
Générer aléatoirement *P2* <= Nbfrag
x ← S(P1)
S(P1) ← S(P2)
S(P2) ← x
Retourne ***S*** *{solution après mutation}*

Fin

Avant de remplacer la nouvelle population on applique l'algorithme PALS sur chaque solution produite par l'opérateur de mutation.

3.5 Algorithme PALS

L'application de l'algorithme PALS se déroule en deux étapes :

Première étape : dans la première étape on détecte, pour chaque solution, ses mouvements possibles, en se basant sur le calcule dedeltaF et deltaC.

Deuxième étape : dans la deuxième étape, on sélectionne un mouvement parmi les mouvements détectés, la sélection d'un mouvement d'une solution se base sur le nombre de contigs de cette dernière. Selon le principe de l'algorithme PALS, pour juger la qualité des solutions, dans le cas d'égalité, il considère le niveau de chevauchement maximal entre les fragments successifs pour parfaire l'évaluation. Dans ce cas, nous sélectionnons la meilleure solution voisine *S'* qui réalise le critère suivant :

$$nContigs(S) < nContigs(S') \text{ ou } (nContigs(S) = nContigs(S') \text{ et } f(S) > f(S')) \quad (4.1)$$

L'algorithme PALSE nécessite comme paramètre un seuil de chevauchement (*cutoff*), donné par l'utilisateur. On a proposé une procédure pour déterminer l'intervalle de ce paramètre (minimum et maximum), car chaque instance a son propre intervalle.

Cette procédure se base sur l'algorithme suivant :

Algorithme 4.11 : max_min_cutoff (chevauch, NbFrag)

```
Chevauch : matrice des chevauchements
NbFrag : nombre de fragments
Début
    minCutoff ← 5000
    maxCutoff ← 0
    pour i ← 1,NbFrag faire
        pour j← 1,NbFrag faire
            si chevauch(i,j) > maxCutoff alors
                maxCutoff ← chevauch(i,j)
            fin si
            si chevauch(i,j) < minCutoff et chevauch(i,j) > 0 alors
                minCutoff ← chevauch(i,j)
            fin si
        fin pour
    fin pour
    rerourn minCutoff
    retourne maxCutoff
    { [minCutoff, maxCutoff] l'interval du cutoff}
fin
```

L'algorithme suivant, nous permet de calculer le nombre de contigs d'une solution.

Algorithme 4.12 : Calcul_nombre_contigs (S, chevauch, cutoff)

```
S : solution
Chevauch : matrice de chevauchements
cutoff : seuil de chevauchement
Début
    Nombre_contigs ← 1 {nombre de contigs de la solution S}
    Pour i ← 1,NbFrag-1 faire
        si chevauch(S(i),S(i+1)) <= cutoff alors
            Nombre_contigs ← Nombre_contigs + 1
        Fin si
    Fin Pour
    Retourne nombre_contigs
Fin
```

Après le calcul de nombre de contigs et la fitness de chaque solution, on passe à l'étape de sélection (deuxième étape), qui est illustrée par l'algorithme suivant :

Algorithme 4.13 : Select_mouvement (liste_mouvement, S)

Liste_mouvement : la liste des mouvements de la solution S
S : solution

Debut

calcul_nombre_contigs (S, chevauch, cutoff)
calcul_nombre_contigs (Liste_mouvement , chevauch, cutoff)
Selecte minimum nombre_contigs

si nombre_contigs(S) = nombre_contigs(Liste_mouvement) ***alors***

Select meilleure fitness

fin si

Retourne S

Fin

3.6 Remplacement

Cette étape consiste à remplacer la population courante par la population des enfants.

4. Expérimentation et Résultats

L'évaluation de la performance de notre assembleur AGPALS, se base sur la comparaison entre la séquence obtenue par AGPALS et la séquence cible (patente), qui se trouve dans le fichier Gen-Frag. Cette évaluation se base sur le score d'alignement et le temps d'exécution.

Score d'alignement : pour avoir une référence (score idéal) d'une séquence, on a fait un alignement de cette dernière avec elle-même.

Pour mieux expliquer, les figures suivantes représentent le score idéal et l'alignement d'une séquence avec elle-même.

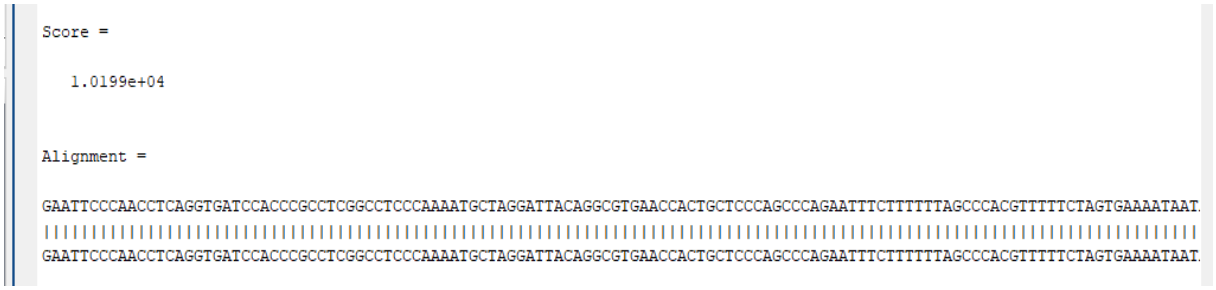


Figure 4.7 : Score idéale de l'instance X60189-4

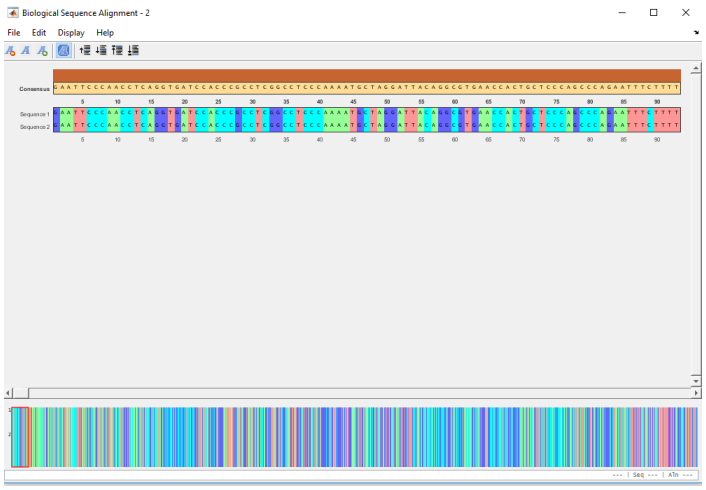


Figure 4.8 : Illustration graphique d'alignement d'une séquence avec elle-même

Le temps d'exécution : le temps d'exécution d'une application est un facteur très important pour l'évaluation, il dépend aussi du matériel utilisé.

Le matériel informatique a toujours une influence sur les résultats d'un programme, dans l'implémentation de notre assembleur, nous avant utilisés un micro-ordinateur portable de la marque '*DELL'* avec les caractéristiques suivants :

- System d'exploitation : *windows 10*
- Processeur : *intel core i5 7^{ième} génération*
- Ram : *8 GB*
- Disque dure : *250 GB*

Le tableau suivant présente les résultats de notre assembleur AGPALS vs un l'assembleur AG avec quelques instances.

L'instance				Assembleur AG		Assembleur AGPALS			Score optimal
Nom d'instance	Nombre de fragments	Nombre de population	Nombre itération	Temps d'exécution	Score obtenu	Temps d'exécution	Score obtenu	cutoff	
X60189-4	39	4	4	12.613921s	5401	125.484748s	5.7293e+03	7	1.0199e+04
						258.642881s	5.7487e+03	200	
		10	4	9.841066 s	6214	558.773439s	5.9153e+03	7	
						466.869417s	5.7507e+03	200	
m15421_5	127	4	4	13.850865s	1.0365e+04	11425.149054s	10437	200	24629
		50	4	31.521246s	1.0561e+04	457407.87654s	1754	7	
		4	30	27.600337s	1.0302e+04	570216.7720s	1673	7	

Tableau 4.2 : Résultats expérimentaux de l'assembleur GA vs l'assembleur AGPALS

Exemple :

Les figures suivantes présentent le résultat d'assemblage de fragments de l'instance X60189-4 comparée avec la séquence cible de cet instance, avec nombre de population =4 et nombre d'itération = 4.

```
Score =

        5401


Alignment =

GATCACCTGAGGTTGGGAATTCGTTGAACCCCAATCTGCCTCTTAGCAAGATCCCCAAGCATGGA-AACGTGCAAAAGAAGCCCGGCTGGTGAGAATATTTTTGTTTTCATGAAGTTGCAGA
||: :|| :|  | :|||:::|| :   :|| |:: || ||: :::||:|||::   ||:||:||:|  || ||| :    |||||:| :  | : ::|::    :  || :| :|| || |:|
GAATTCCCAACCTCAGGTGATCCACCCGCCTCGGCCTCCCAAAATGCTAGGATTACAGGCGTGAACCAC-TGC-TCC-CAGCCCAGAATTTCTTTTTTAGCCCACGTTTTTCTAG-TGAAAA

Elapsed time is 12.613921 seconds.
>>
```

Figure 4.9 : Résultat d'assembleur GA avec l'instance X60189-4

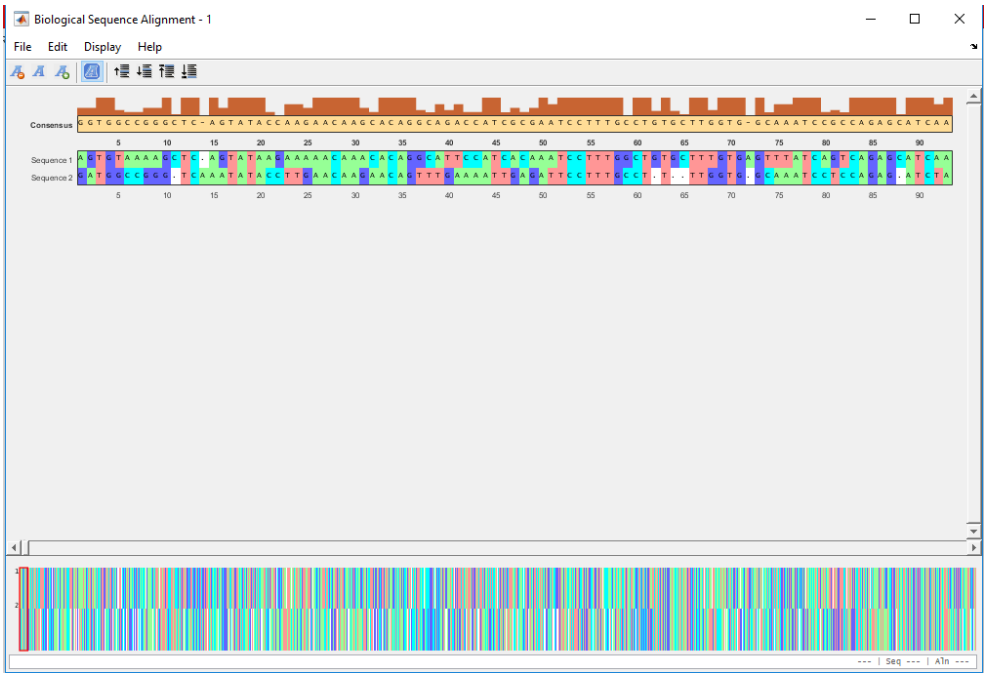


Figure 4.10 : illustration graphique d'alignement

La figure 4.9 montre l'alignement de ces deux séquences (résultat et cible), le score d'alignement et le temps d'exécution. Sachant que la valeur du score indique la précision de l'assemblage. Si la valeur de score s'approche de la valeur du score idéal, l'assembleur est optimal.

5. Conclusion

Dans notre projet d'assemblage des fragments d'ADN avec l'algorithme génétique AG et l'algorithme AGPALS, nous avant conclu que la performance d'un assembleur d'ADN conçu par un algorithme génétique, dépend de la taille de la population (si on augmente la taille de population on arrive en un bon résultat), et nombre d'itérations (si on augmente le nombre d'itérations, on obtient un score d'alignement grand).

Alors que la performance de l'assembleur AGPALS dépend du seuil de chevauchement (*cutoff*).

L'hybridation de l'algorithme PALS avec l'algorithme génétique (AG) améliore les résultats de l'assemblage (score d'alignement), mais augmente le temps d'exécution.

Conclusion générale

L'assemblage de fragments d'ADN est un problème issu du domaine de la bioinformatique, et se pose au niveau du séquençage des génomes. Ce problème consiste à construire une séquence d'ADN à partir de plusieurs centaines (voire des milliers) de fragments obtenus par les biologistes du laboratoire. Il est connu comme étant un problème d'optimisation combinatoire très difficile, car le temps de trouver un ordre total des fragments donnés, pour former la séquence parent, augmente de façon exponentielle lorsque le nombre de ces fragments et leurs tailles augmentent aussi.

Dans ce mémoire, nous avons résolu, ce problème, par une hybridation de l'algorithme génétique (AG) avec l'algorithme PALS pour augmenter la robustesse du processus de recherche des bonnes solutions et améliorer le processus d'exploration et d'exploitation.

Ces caractéristiques sont démontrées par les résultats expérimentaux qui sont faites sur des différentes instances de fragments d'ADN et qui montrent que l'algorithme hybride AGPALS trouve de bonnes solutions.

Pour traiter ce thème, nous avons fait une étude détaillée sur le problème d'assemblage de fragments d'ADN et l'algorithme génétique qui ont été l'objet de deux premiers chapitres. Nous avons présenté la conception et l'implémentation de l'algorithme hybride AGPALS dans le troisième et quatrième chapitre.

Une étude expérimentale présentée, dans le quatrième chapitre, a montré que l'hybridation de l'AG avec PALS a donné de bons résultats par rapport à ceux de l'AG. Mais pour améliorer le temps de traitement des grandes instances et assurer très rapidement sa convergence, nous envisagerons une ré-conception de l'AGPALS pour l'implémenter et l'exécuter sur des machines parallèles, ceci reste comme une perspective.

Bibliographie

- [1] M. Pop, Shotgun sequence assembly. Advances in Computers, 2004.
- [2] Michael L. Engle, Christian Burks, GenFrag 2.1: new features for more robust fragment assembly benchmarks, *Bioinformatics*, Volume 10, Issue 5, September 1994, Pages 567–568, https://doi.org/10.1093/bioinformatics/10.5.567
- [3] E. Alba et G. Luque, A new local search algorithm for the DNA fragment assembly problem, Dans Carlos Cotta et Jano van Hemert, editeurs, Evolutionary Computation in Combinatorial Optimization, April 11 13 2007.
- [4] L. Excoffier et D. Roessli, Origine et évolution de l'ADN mitochondrial humain: le paradigme perdu, Année 1990 2-1 pp. 25-41
- [5] Poirier, N. Nicole, L'utilisation de la preuve par l'ADN et ses impacts sur notre société, 2014.
- [6] A. Boudet, L'ADN électromagnétique et la communication entre cellules, 1 septembre 2011- version augmentée 30 avril 2014, https://www.spirit-science.fr/doc_humain/ADN6photons.html.
- [7] G. Camus, Les rôles de l'ATP, date de publication : jeudi 23 juin 2011, dernière modification : lundi 11 décembre 2017.
- [8] S. Bourgoin, Protocoles rapides pour la préparation d'ADN à partir d'échantillons caractéristique de la biologie judiciaire, université du Québec à Trois-Rivières, décembre 2000.
- [9] MJ. Levene, J. Korlach , SW. Turner, M. Foquet et HG. Craighead H, Zero-mode waveguides for single-molecule analysis at high concentrations, 2003 Jan 31;299(5607):682-6.
- [10] P.A Brailiard**,** Enjeux philosophiques de la biologie des systèmes, Université Paris 1 – Panthéon Sorbonne, Ecole Doctorale de Philosophie, Octobre 2008.
- [11] M. Delarue et G. Furelaud , Le séquençage d'un ADN, Date de publication : Samedi 1 mai 2004, Dernière modification Mardi 20 mars 2018
- [12] N. Charrat, Diagnostique moléculaires et application en agroalimentaire : caractérisation D des souches de campylobacter isolées à partir du poulet de chair au maroc, Université Mohammed V, Rabat, N° d'ordre : 2987, 19 mai 2017.

- [13] J. Sengenès, Development de methods de séquencage de seconde generation pour l'analyse des profiles de methylation de l'AND, Université Paris VI, 30 Mars 2012.
- [14] S. Corem, Le séquençage à haut debit, Université Pierre et Marie Curie, Juin 2012.
- [15] C. SAAD, caractérisation des erreurs de séquençage non aléatoires, Université de Lile, Septembre 2018.
- [16] A. Ben Ali, Contributions à la résolution de problème d'optimisation combinatoire NP-difficile, Universite Mohamed Khider Biskra, 19 Avril 2018.
- [17] P. Fouilhoux, Optimisation Combinatoire: Programmation Linéaire et lgorithmes, Université Pierre et Marie Curie, 29 septembre 2015.
- [18] L Belhoul, Résolution de problèmes d'optimisation combinatoire mono et multi-objectifs par énumération ordonnée, Université Paris-Dauphine, 13 Décembre 2014.
- [19] S. Ben Ismail, Introduction à l'optimisation combinatoire, 2012.
- [20] P. Lacomme et C. Prins, Algorithmes de graphes, Groupe Eyrolles, 1994, 2003, ISBN : 2-212-11385-4, 2e édition 2003
- [21] A. Ben Ali, G Luque, E Alba et K E. Melkemi, An improved problem aware local search algorithm for the DNA fragment assembly problem. Soft Computing, 21(7), 2017.
- [21] : L. Djerou, Algorithmes génétiques, Support de cours du module AAO - Algorithmes Avancés et optimisation, université Mohamed Khider BISKRA,2018.
- [22] T. Vallée et M. yilsizoglu, Présentation des algorithmes génétiques et de leurs applications en économie, Université de Nantes LEA-CIL, Université Montesquieu Bordeaux IV, 7 septembre 2001, v. 1.2
- [23] L. Djerou, Codage réel et les opérateurs génétiques, Support de cours du module AAO -Algorithmes Avancés et optimisation, université Mohamed Khider BISKRA, 2018.
- [22] https://www.ncbi.nlm.nih.gov/