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 :…./M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : Réseaux et Technologies de l'Information et de la Communication(RTIC)

Traitement parallèle pour l'assemblage de fragments d'ADN

Par :

HASSANI NOUR ELHOUDA

Soutenu le../../…. devant le jury composé de :

Nom Prénom	grade	Président
Moussaoui Manel	grade	Rapporteur
Nom Prénom	grade	Examinateur

Année universitaire 2020-2021

Table des matières

Liste des figures

Introduction général

Introduction général

Le problème d'assemblage de fragments d'ADN est un problème issu du domaine de la Bioinformatique, et de rechercher un ordre total se pose au niveau du séquençage des génomes. Le séquençage d'ADN est devenu une pratique indispensable dans les domaines de médecine, de phylogénétique et du crime. Il consiste à construire une séquence d'ADN à partir de plusieurs centaines (voire des milliers) de fragments obtenus par les biologistes du laboratoire.

L'émergence de ce problème dans la bioinformatique a été la conséquence du besoin d'automatiser le séquençage de séquence de l'ADN de taille considérable ce problème classé comme un problème NP-Difficile [Pevener 2006] pour n fragments, il y a 2^nn! Configurations possibles, pour résoudre ce problème nous avons appliqué l'algorithme PALS (de l'anglais Problème Aware Local Search) est une méthode de recherche locale plus rapide et plus compétitif, développe par Alba et Luque (Alba et Luque, 2007) pour résoudre ce problème. [1]

Le principe de travail de l'algorithme de recherche local (PALS) est de chercher itérativement une solution optimale ou une solution quasi optimale, et puisque le temps d'exécution de l'algorithme est augmenté avec la taille de la base des fragments,
Pour cette raison, PALS nécessitent une parallélisassions pour minimiser le temps d'exécution.

La parallélisassions est faite par l'attribution de différentes parties de l'espace de recherche à différents processeurs. Pour la satisfaction de l'utilisateur qui recherche toujours des résultats efficaces dans un temps réduit,

Dans ce projet notre objectif est de proposer une version parallèle de l'algorithme de recherche local (PALS) nommé (PPALS) pour minimiser le temps de calcul.

Ce mémoire débute par une introduction générale qui présente la problématique. Le mémoire est composé à trois chapitre le premier qui présent la analyse d'ADN et les stratégies de séquençage et les approches d'assemblage. Le deuxième chapitre présente la recherche locale et l'algorithme PALS. Dans le dernier chapitre nous allons présenter la conception et l'implémentation de notre projet.

Chapitre I :

Assemblage de fragment d'ADN

1. Introduction

La biologie est un domaine scientifique nécessaire et complexe. La collaboration de mathématicien et physicien, biologiste, informaticien, est la base de la bio-informatique Les fruits de cette association sont l'analyse du génome et le séquençage du génome. Dans ce travail, nous allons nous intéresser au séquençage du génome et le problème d'assemblage de fragment d'ADN par ce que l'ADN est sensibles aux séquences .De ce fait, appliquée quelques approches d'assemblages pour mieux adaptation d'assemblage.

2. Définition bioinformatique

La bioinformatique tire des connaissances de l'analyse informatique des données biologiques. Ceux-ci peuvent se composer des informations stockées dans le code génétique, mais aussi des résultats expérimentaux provenant de diverses sources, des statistiques des patients et de la littérature scientifique. La recherche en bioinformatique comprend le développement de méthodes pour le stockage, la récupération et l'analyse des données. La bioinformatique est une branche en plein développement de la biologie et est très interdisciplinaire, utilisant des techniques et des concepts de l'informatique, des statistiques, des mathématiques, de la chimie, de la biochimie, de la physique et de la linguistique. Il a de nombreuses applications pratiques dans différents domaines de la biologie et de la médecine.[2]

3. Définition d'ADN

L'ADN ou acide désoxyribonucléique est le matériel où sont stockés l'information génétique et l'ensemble des caractères héréditaires d'une cellule. Sa structure moléculaire est similaire à une échelle constituée de quatre sortes de barreaux. Elle est représentée par une double chaîne de nucléotides de forme hélicoïdale (figure 1). L'ADN est constitué à partir de quatre sous unités (nucléotides) qui contiennent les bases azotées adénine (A), guanine (G), cytosine (C) et thymine (T). Les bases s'assemblent selon une complémentarité exclusive: A s'apparie uniquement avec T, et G avec C. Le rôle fondamental de l'ADN est de stocker l'information génétique. L'ADN peut être vu comme le matériel ("hardware") dépositaire des caractères héréditaires. C'est la mémoire du code génétique des êtres vivants. Le matériel génétique de l'ADN peut être reproduit par une opération de réplication. Il peut être traité en vue d'élaborer de nouvelle molécules nécessaires au métabolisme des cellules durant les opérations de transcription et de traduction. Il est représenté textuellement comme une séquence orienté de lettres : 5' AGGCTT… 3'.[3]

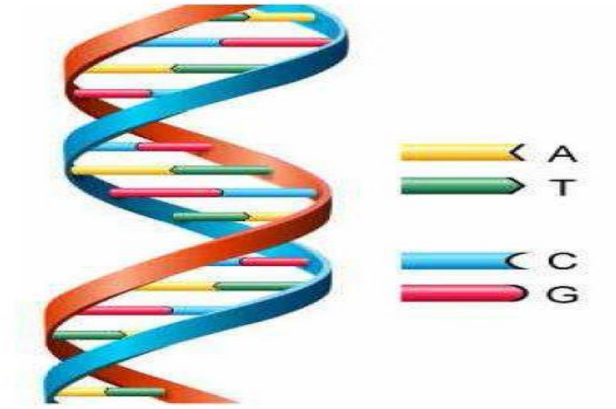


Figure 1.1 : Représentation schématique de la double hélice de l'ADN [4]

3.1. L'analyse d'ADN

L'analyse d'ADN est devenue de nos jours une pratique indispensable de la police technique et scientifique. Elle permet par exemple d'identifier des victimes, d'établir la présence de suspects sur le lieu du crime ou de vérifier le lien de parenté entre deux personnes. Elle consiste à comparer des segments choisis de molécules d'ADN de différents individus afin de fournir des indices où il n'y a pas de témoins. Elle est également utilisée pour trouver des traitements à des maladies génétiques. En biologie, l'analyse d'ADN est devenue un outil important pour la classification des espèces.
L'objectif principal de l'analyse d'ADN est de déterminer la séquence complète du génome et de son contenu génétique.[5]

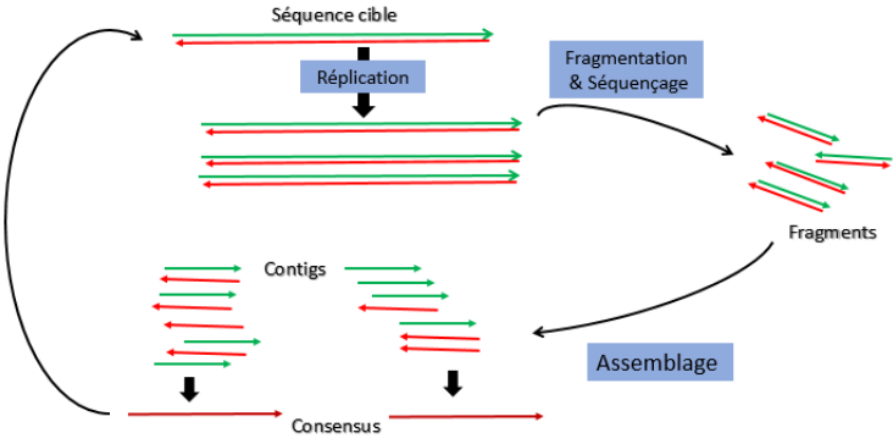


Figure 1.2 : Processus d'analyse d'ADN

3.2. Les étapes de l'analyse d'ADN

L'analyse d'ADN passe par 6 étapes : [5]
- Duplication de la séquence d'ADN : création de plusieurs copies identiques de la séquence d'ADN.
- Sonification : découpage aléatoire de l'ADN en petits fragments.
- Séquençage : consiste à déterminer l'ordre d'enchaînement des nucléotides.
- Appel des bases : la lecture des bases A, T, C et G à partir des résultats de séquençage (électrophorégramme) ;
- Agencement (layout) : positionnement des fragments selon les chevauchements calculés ;

- Consensus : détermination de l'ADN final.

Les trois premières étapes de séquençage de l'ADN se produisent dans le laboratoire où de grands fragments d'ADN sont dupliqués puis fragmentés en petits fragments qui sont séquencés individuellement (méthode de Sanger).

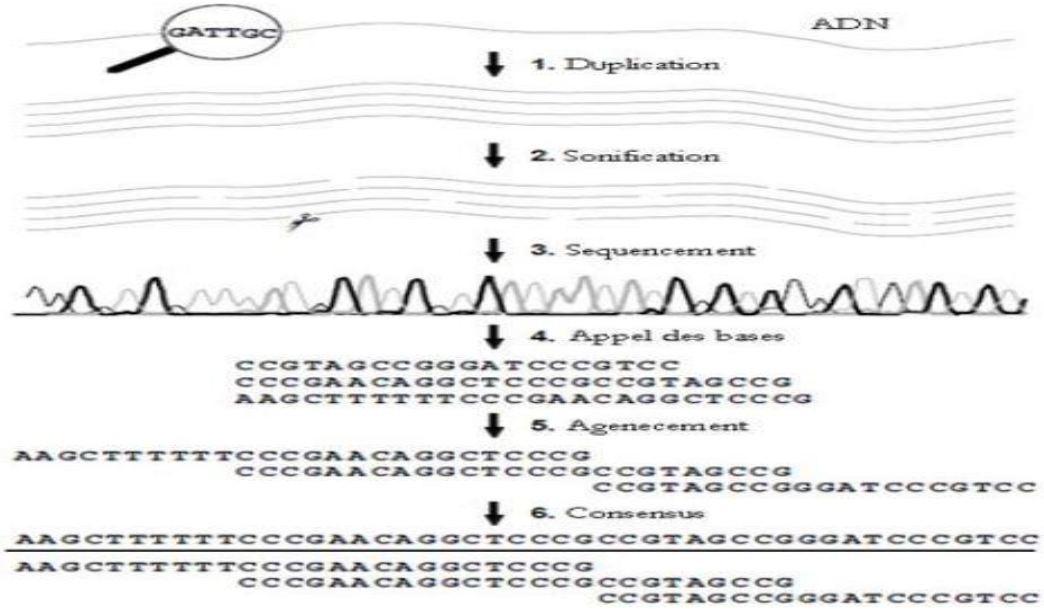


Figure 1.3: Les étapes de l'analyse d'ADN [5]

4. Séquençage de fragments d'ADN

Séquencer un brin d'ADN, revient à lire l'enchaînement, ou séquence, des nucléotides constitutifs de cette molécule. En effet, on détermine la succession des bases nucléiques. Les méthodes de séquençage actuelles ne permettent pas de séquencer un grand nombre de bases. Or la séquence du génome humain comprend environ 3,2 milliards de paires de bases. Il sera donc impossible de séquencer d'un coup la totalité du génome. Les biologistes sont incapables de manipuler des molécules d'ADN d'un million de bases. Pour obtenir suffisamment de séquences chevauchantes et pour réduire les erreurs de séquençage, il faut atteindre un certain niveau de redondance, c'est-à-dire produire une quantité de séquences aléatoires représentant plusieurs fois la longueur de la séquence d'intérêt. Ceci conduit à un nombre très important de séquences à réaliser. Si par exemple 10 copies de la séquence d'ADN sont disponibles, chaque base de la séquence cible a été lue 10 fois en moyenne. Même à 10X, des "trous" peuvent donc subsister, laissant la séquence finale très légèrement incomplète. [5]

Le processus de séquençage d'un génome est basé sur la fragmentation aléatoire pour obtenir des morceaux d'ADN, de quelques paires de bases, faciles à manipuler. Ces petits fragments sont alors séquencés individuellement. La séquence complète du génome est reconstruite à partir de ces séquences unitaires sur la base des chevauchements. Les

fragments ayant une grande partie chevauchante sont plus d'ADN dont elles dérivent ont une partie de leur longueur en commun. La fragmentation étant aléatoire, les molécules d'ADN de l'échantillon ne sont pas toutes fragmentées aux mêmes endroits. [5]

4.1. Les stratégies de séquençage

4.1.1. La méthode hiérarchique ou "clone par clone"

Le génome est découpé en un nombre "restreint" (quelques dizaines de milliers) de fragments de grande taille (50 à 200 kilo paires de base) qui couvrent l'ensemble du génome. Ces fragments sont clonés dans des vecteurs spéciaux : les YAC ("Yeast Artificial Chromosome" - problème d'échange de fragments d'ADN), les BAC ("Bacterial Articifial Chromosome") ou des vecteurs dérivés du phage P1 (les PAC).

Une carte physique des clones est établie pour faciliter l'obtention de la séquence finale du génome : elle permet d'ordonner les clones dans le génome. Les cartes de liaison disposent des marqueurs ordonnés le long des chromosomes par la mesure de leur liaison deux à deux. Ces cartes de liaison permettent de se repérer dans le génome et sont une aide essentielle dans la construction de la carte physique.

Un sous-ensemble avec un minimum de recouvrement (pour avoir une couverture la plus complète possible du génome) est ensuite choisi et séquencé en "vrac" : chaque clone de grande taille est découpé en un grand nombre de fragments de petite taille (environ 2000 paires de bases) et les extrémités sont séquencées individuellement.

Les problèmes d'assemblage ne se posent qu'à l'échelle des grands fragments et sont facilement résolus en multipliant le nombre de lectures dans ces zones. [6]

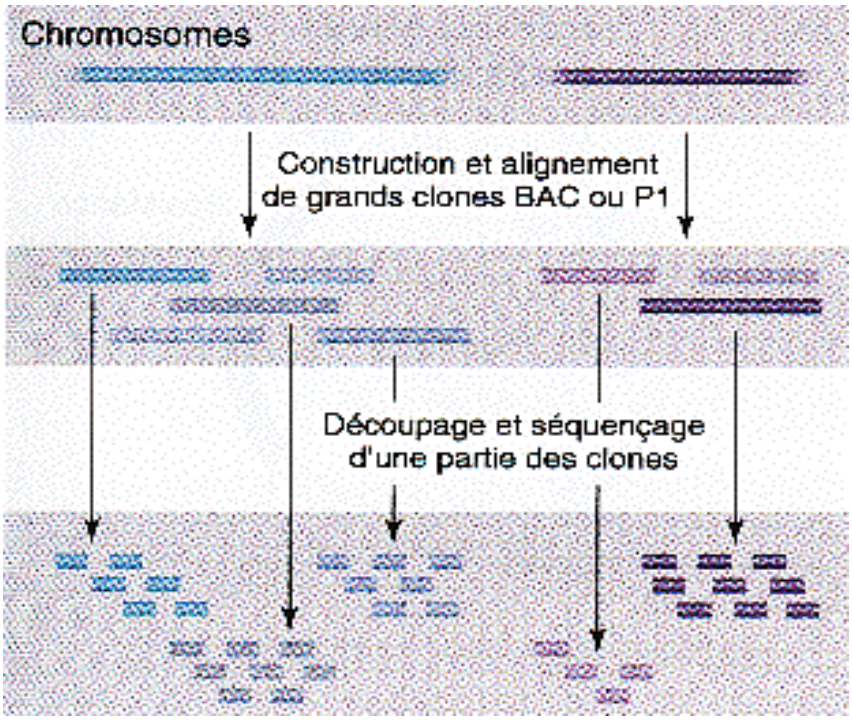


Figure 1.4: La méthode hiérarchique [6]

4.1.2. La méthode de séquençage aléatoire global ou "*shotgun*"

Elle a été utilisée massivement pour le séquençage d'un grand nombre de génomes notamment bactériens (séquençage *de novo*). Schématiquement, cette méthode consiste à fragmenter le génome entier à étudier en petits fragments d'ADN à l'aide de moyens mécaniques. Sur chaque fragment, une réaction de ligation permet de fixer de courtes séquences d'ADN appelées adaptateurs, ces derniers servant d'amorce pour la PCR. Ces fragments sont ensuite intégrés dans des plasmides et constituent une bibliothèque (*library*) de fragments aléatoires d'ADN simple brin. Ils sont ensuite amplifiés par PCR, par exemple, puis séquencés à l'aide de la méthode de Sanger. À l'aide de logiciels informatiques, les séquences sont ensuite alignées et recoupées par chevauchement. Ces séquences représentent plusieurs blocs de séquences continus : on parle alors de « contig ». Certaines séquences manquent : ce sont des trous de séquences (*gap*). Ceux-ci sont comblés par séquençage à partir des séquences déjà déterminées. À partir de ces dernières, sont dessinées des amorces de PCR servant pour l'amplification en direction de ces trous. Cette technique nécessite de séquencer le génome cible plusieurs fois (minimum cinq fois), tant pour éviter les erreurs de séquençage que pour s'assurer du maximum de chevauchement entre les séquences et faciliter l'ordonnancement des séquences. Cette technique est essentiellement utilisée pour des génomes de petite taille comme ceux des bactéries.[7]

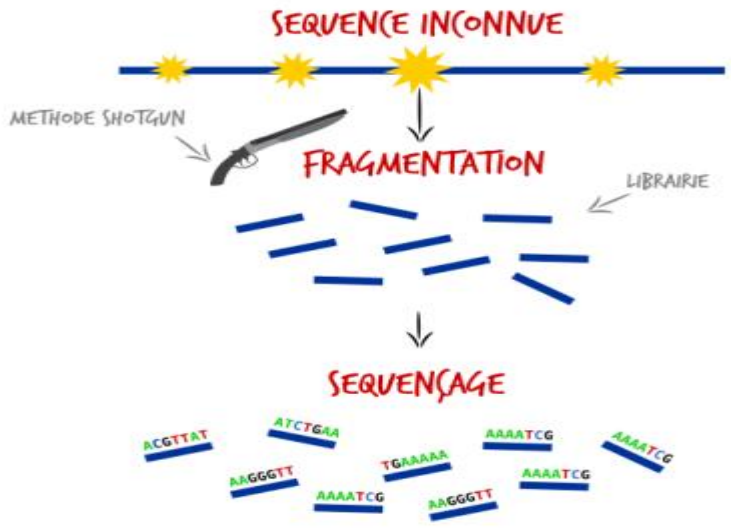


Figure 1.5 : La méthode de séquençage shotgun[6]

5. Problèmes d'assemblage des fragments d'ADN

Le séquençage des fusils de chasse Cette méthode a été largement utilisée pour la détermination de l'ADN Séquences biologiques (POP 2004).Premièrement, précision multiple Dupliquer la séquence d'ADN originale. Et chaque La copie est divisée aléatoirement en plusieurs morceaux assez courts Lu par le séquenceur. La prochaine étape est Assemblez toutes les opérations de lecture dans la séquence de croissance. Elle comporte trois volets: Différentes étapes: chevauchement, mise en page, puis consensus.[1]

La première étape consiste à trouver tous les chevauchements Entre les morceaux. Pour ce faire, nous avons comparé toutes les possibilités Pour déterminer leur similitude. Pour chaque

couple, Nous calculons la correspondance la plus longue entre le premier suffixe Préfixe du fragment et du second.

En général, dynamique L'algorithme de programmation de l'alignement semi - global est introduit Utilisez cette étape pour obtenir des points de chevauchement.[1]

Mise en page cette étape consiste à déterminer Schéma de fragmentation basé sur le chevauchement calculé Fractions, c'est - à - dire l'ordre dans lequel tous les fragments sont conservés le plus Contraintes de chevauchement. C'est l'étape la plus compliquée Parce qu'il est difficile de détecter un véritable chevauchement De nombreux défis. Un défi important est La séquence originale est découpée au hasard en plusieurs Fragments courts, direction perdue. C'est impossible. Déterminer les torons (à partir de deux supports opposés) (de l'ADN) La source d'un fragment particulier. Pour Obtenez des points de chevauchement et essayez chaque direction possible Chaque paire de fragments. Un autre défi important est En fait, la couverture n'est pas complète. Quand L'algorithme assembleur ne peut pas se connecter à l'ensemble de fragments Ça devient une contusion. La contusion est une disposition dans laquelle tout Le chevauchement entre les segments consécutifs est supérieur à Seuil (paramètre de coupure).[1]

Le but de cette étape est de produire de l'ADN Séquence de mise en page des fragments. À cette étape, un multiple Construire un alignement de séquence de fragments pour chaque fragment Pour créer une véritable réflexion Un brin du brin d'ADN de la mère. Le plus fréquent Les techniques utilisées à cette étape sont les suivantes: Parvenir à un consensus. Le processus d'étalonnage doit tenir compte: Erreur d'appel de base lors de la lecture de l'ADN Séquence.[1]

La qualité du consensus peut être mesurée par des observations Répartition de la couverture. Mesure de la couverture Redondance des données et Fragment d'un nucléotide spécifique dans l'ADN cible La séquence apparaîtra. Il est calculé comme la somme La longueur du fragment dépasse la longueur de la cible Séquence d'ADN

$$\boldsymbol{Couverture} = \frac{\sum_{i=0}^{k} \boldsymbol{Longueur\ du\ fragment\ i}}{\boldsymbol{Longueur\ de\ la\ séquence\ cible}}$$

Où K est le nombre de fragments. Plus la couverture est élevée, Plus l'écart est petit, plus les résultats sont bons. Cet article traite principalement des problèmes de mise en page. C'est aussi connu sous le nom de problème d'assemblage de fragments. Ce problème peut être considéré comme un problème d'optimisation combinatoire Question. Le but réel est de trouver un arrangement Réduire la quantité de débris en contact, Pour atteindre un but contusionné. De nombreux Assembleur utilisant une fonction objective favorable à la mise en page Il y a un fort chevauchement entre les segments consécutifs. [1]

6. Les approches d'assemblage

Il y a deux types grands de techniques d'assemblage de séquences qui peuvent être utilisées: ensemble de novo et comparatif.

6.1. L'ensemble de novo

Pour assembler des lectures de séquençage, nous utilisons des logiciels appelés assembleurs. Les assembleurs, à l'aide d'algorithmes variés, utilisent les lectures et tentent de les agencer de façon à obtenir la séquence complète de la molécule séquencée. Lorsque plusieurs lectures sont assemblées, elles forment ce qu'on appelle un « contig ». Il s'agit essentiellement d'une portion de la séquence finale que nous cherchons à assembler, et donc de la molécule d'ADN ou du génome séquencé.
La plupart des algorithmes d'assemblage de novode génomes couramment utilisés appartiennent à deux grandes familles. [8]

6.1.1. Assembleur de type Overlap-Layout-Consensus

La première famille, celle des algorithmes de type Overlap-Layout-Consensus (OLC), a été élaborée pour la première fois en 1980. Elle est mieux adaptée à l'assemblage de longues lectures.

Les algorithmes OLC utilisent une approche assez simple et instinctive qui est très modulaire. Les étapes de cette approche est :

1- Calculer tous les chevauchements possibles entre toutes les paires de lectures fournies
2- Créer un graphe dont les nœuds représentent chacune des lectures.
3- Deux nœuds reliés entre eux correspondent à deux lectures qui se chevauchent
4- . L'assembleur cherche ensuite un chemin qui traverse le graphe en passant par chaque nœuds une seule fois
5- Les contigs sont construits au fur et à mesure que les nœuds sont sélectionnés et que leurs lectures respectives sont ajoutées à la séquence consensus de l'assemblage.
6- En ne sélectionnant que les chemins qui passent une seule fois par chaque nœud, on s'assure de ne pas intégrer la même lecture dans plusieurs contigs .
7- Cette étape correspond à chercher un chemin Hamiltonien dans un graphe.
8- Les chemins divergents peu couverts (engendrés par des lectures erronées) sont retirés pour simplifier la résolution de l'assemblage et le risque de commettre des erreurs.[8]

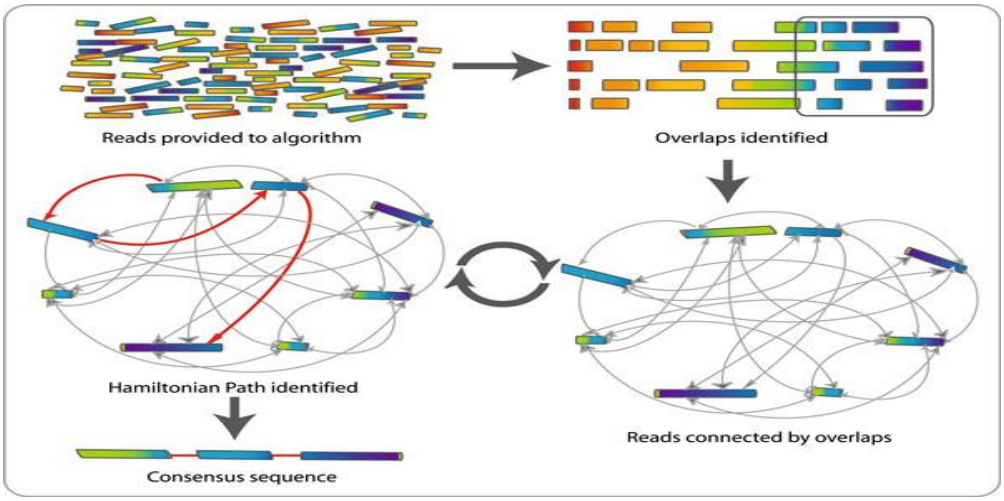


Figure 1.6 : Schéma de la méthode OLC (Overlap-Layout-Consensus)

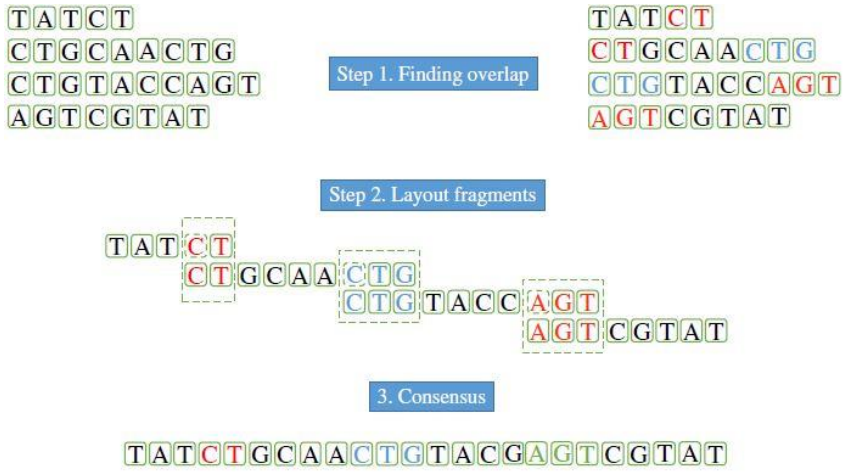


Figure 1.7 : Exemple de la méthode OLC (Overlap-Layout-Consensus)[9]

6.1.2. Assembleur de type graphe DeBruijn

La deuxième famille d'algorithmes d'assemblage est celle des assembleurs de type graphe DeBruijn (DBG). Cette méthode, qui est mieux adaptée aux courtes lectures, a été présentée en 2001 avec le logiciel Euler. Le principe derrière les assembleurs DBG vise à simplifier la recherche d'un chemin Hamiltonien dans le graphe en le transformant en recherche d'un chemin Eulérien. Un chemin Eulérien est un chemin qui passe à travers tous les arcs d'un graphe une seule fois. Contrairement à la recherche d'un chemin Hamiltonien qui est NP-complet, la recherche d'un chemin Eulérien peut se faire en temps polynomial.[8]

L'utilisation de graphes DeBruijn permet l'utilisation de nombreux algorithmes pour facilite la recherche de chemins Eulériens ce qui simplifie grandement plusieurs étapes de résolution du graphe . Par contre, un des paramètres essentiels à une bonne utilisation des graphes DeBruijn, la taille de k-mère, n'est pas aussi facile à manier. La taille de k-mère choisie change grandement l'allure du graphe produit. En supposant qu'il existe une taille optimale de k-mère que l'on décrira comme « k » qui est un nombre impair (pour éviter qu'un k-mère soit également son propre complément inverse) positif forcément plus petit que la taille de nos lectures, choisir une valeur de k-mère très inférieure ou très supérieure à k pourrait devenir nuisible à l'assemblage. Si une valeur de k-mère trop petite est choisie, le graphe résultant sera difficile à résoudre, une taille de k-mère petite signifiera que les k-mères générés seront probablement très redondants, par conséquent de nombreux arcs quitteront et arriveront de chacun des nœuds du graphe, produisant un nombre important de cycles dans le graphe . Tout cycle à l'intérieur du graphe DeBruijn est le signe de la présence d'une répétition, puisqu'il existe un arc partant du nœud courant vers un nœud déjà visité.[8]

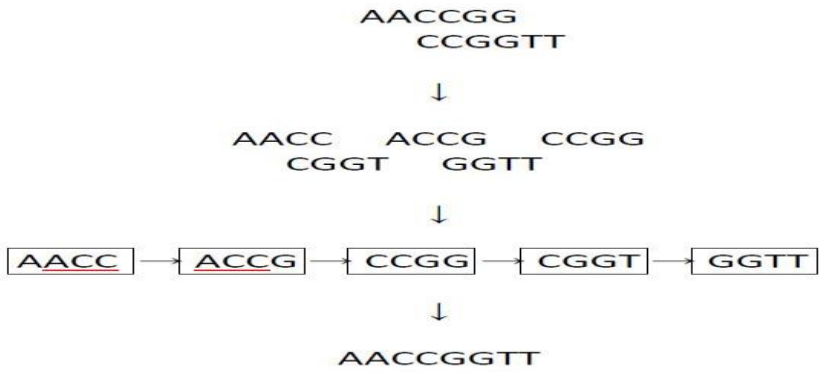


Figure 1.8: Représentation schématique de l'assemblage de deux séquences par la méthode graphique « De Bruijn»[10]

6.2. L'ensemble de comparatif (cartographie)

Le comparatif, ou la cartographie, ensemble est employé pour les génomes qui ont une séquence existante, ou a un génome qui est assimilé à un autre organisme qui a déjà un génome assemblé, qui peut être employé comme référence. [11]

7. Conclusion

Dans ce chapitre, nous avons présenté des concepts en biologie computationnelle et bioinformatique et analysé le problème de l'assemblage de fragments d'ADN par des détails qui varient en fonction des caractéristiques de la séquence du génome et des différentes technologies utilisées.

Les chapitres de la partie II parlent en détail des solutions d'assemblage étudiées, Algorithme génétique (PALS).

Chapitre II:

Algorithme de recherche local PALS

1. Introduction

Dans ce chapitre nous allons parler sur les heuristiques qui sont des méthodes de résolution et stratégie de bon sens purement algorithmiques qui permettent d'obtenir des solutions à n'importe quel problème décisionnel rapidement, Ensuite, nous présentons une métaheuristique ainsi que quelques notions fondamentales comme recherche locale.

L'approche de la recherche locale est très efficace pour résoudre les problèmes NP-difficile ou NP-complet surtout pour ceux ayant un espace de recherche avec un voisinage de très grande taille, elle garantit une solution en temps abordable grâce à sa rapidité et à son principe relativement simple. Dans certain cas, la qualité de la solution trouvée n'est pas bonne.[5]
Ce chapitre est consacré à la notion de la recherche locale. Nous ferons en particulier une description de l'algorithme PALS.

2. Méthodes heuristiques

Les méthodes heuristiques sont utilisées pour des problèmes de grandes tailles pour lesquels les méthodes exactes prennent un temps exponentiel. Généralement, on n'obtient pas la solution optimale mais une solution approchée, assez bonne. Une méthode heuristique se déplace dans l'espace de solutions, de voisin en voisin, en se basant seulement sur les informations locales (la solution courante et son voisinage), avec objectif d'atteindre un optimum au-delà duquel aucun mouvement local n'est possible. Cette méthode de par sa simplicité est caractérisée par une moindre consommation de mémoire, mais elle ne dirige pas généralement la résolution du problème vers un optimum global. .[13]

Les heuristiques sont des méthodes qui exploitent la structure d'un problème considéré afin de trouver une solution approchée de bonne qualité en un temps raisonnable, *i.e.* aussi faible que possible. En d'autres termes, elles permettent généralement de trouver une solution approchée à un problème de classe NP en un temps polynomial. Ces heuristiques peuvent être déterministes ou stochastiques.
Les heuristiques les plus simples qu'on retrouve dans la littérature sont les heuristiques gloutonnes (DEVORE et TEMLYAKOV [1996]). Ce type d'heuristique fait une succession de choix en fonction de certaines caractéristiques (Exemple : choisir la ville la plus proche parmi un ensemble de villes encore non parcourues), jusqu'à ce que la solution soit réalisable, et cela, sans retour en arrière possible.
Bien que le principe soit assez générique, il doit cependant être adapté en fonction du problème, car ces heuristiques sont généralement guidées par des caractéristiques spécifiques au problème considéré et en sont donc totalement dépendantes.[14]

3. Méthodes métaheuristiques

Les métaheuristiques sont des heuristiques adaptées à chaque problème, qui ont été misent au point, sans changements majeurs dans l'algorithme, afin de résoudre des problèmes d'optimisation difficile, pour lesquels nous ne connaissons pas de méthodes classiques plus efficaces. Ces méthodes sont, pour la plupart, inspirées de la biologie (algorithmes évolutionnaires) ou de l'éthologie (essaims particulaires, colonies de fourmis).[15]

Un métaheuristique est formellement défini comme un processus de génération itérative qui guide un heuristique subordonné en combinant des concepts intelligemment différents pour

explorer et exploiter l'espace de recherche, des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver des solutions efficaces presque optimales[16]

Les méta-heuristiques sont des méthodes généralisées pour la résolution de problèmes NP. Ainsi, à l'inverse des heuristiques, elles sont indépendantes du problème considéré. De plus, des preuves de convergence vers la solution optimale ont été établies (FAIGLE et KERN [1992]) montrant que la probabilité de trouver la solution optimale augmente si on laisse un temps infini. Bien qu'en pratique il n'est pas possible de laisser un temps infini, les méta-heuristiques ont montré de très bonnes performances sur un temps fini. [14]

4. Problème NP

Est un problème connu comme étant NP et si une solution au problème est connue, la démonstration de l'exactitude de la solution peut toujours être réduite à une vérification P (temps polynomial). Si P et NP ne sont pas équivalents, la solution des problèmes de NP nécessite (dans le pire des cas) une recherche exhaustive.[18]

Soit avec une méthode exacte pour chercher l'optimum global, soit avec une méthode approchée pour chercher une solution de bonne qualité dans un délai raisonnable. Une résolution avec une méthode exacte pour un problème NP-difficile est souvent couteuse car le temps d'exécution est non polynomial en la taille des instances à traiter. Si les instances du problème sont de petites tailles, la résolution avec une méthode exacte reste envisageable. Par contre, si les instances sont de grandes tailles, une méthode de résolution approchée reste l'unique façon pratique de résoudre le problème. Le principe d'une telle approche est une exploration de l'espace des solutions a la recherche d'une bonne solution (pas forcement l'optimale) en un temps d'exécution raisonnable.[17]

Un problème est attribué à la classe NP (temps polynomial non déterministe) si elle est résolue en temps polynomial par une machine de Turing non déterministe.[18]

5. Classification des métaheuristiques

Les métaheuristiques peuvent être classées de nombreuses façons. Ce diagramme tente de présenter où se placent quelques-unes des méthodes les plus connues. Un élément présenté à cheval sur différentes catégories indique que l'algorithme peut être placé dans l'une ou l'autre classe, selon le point de vue adopté.[19]

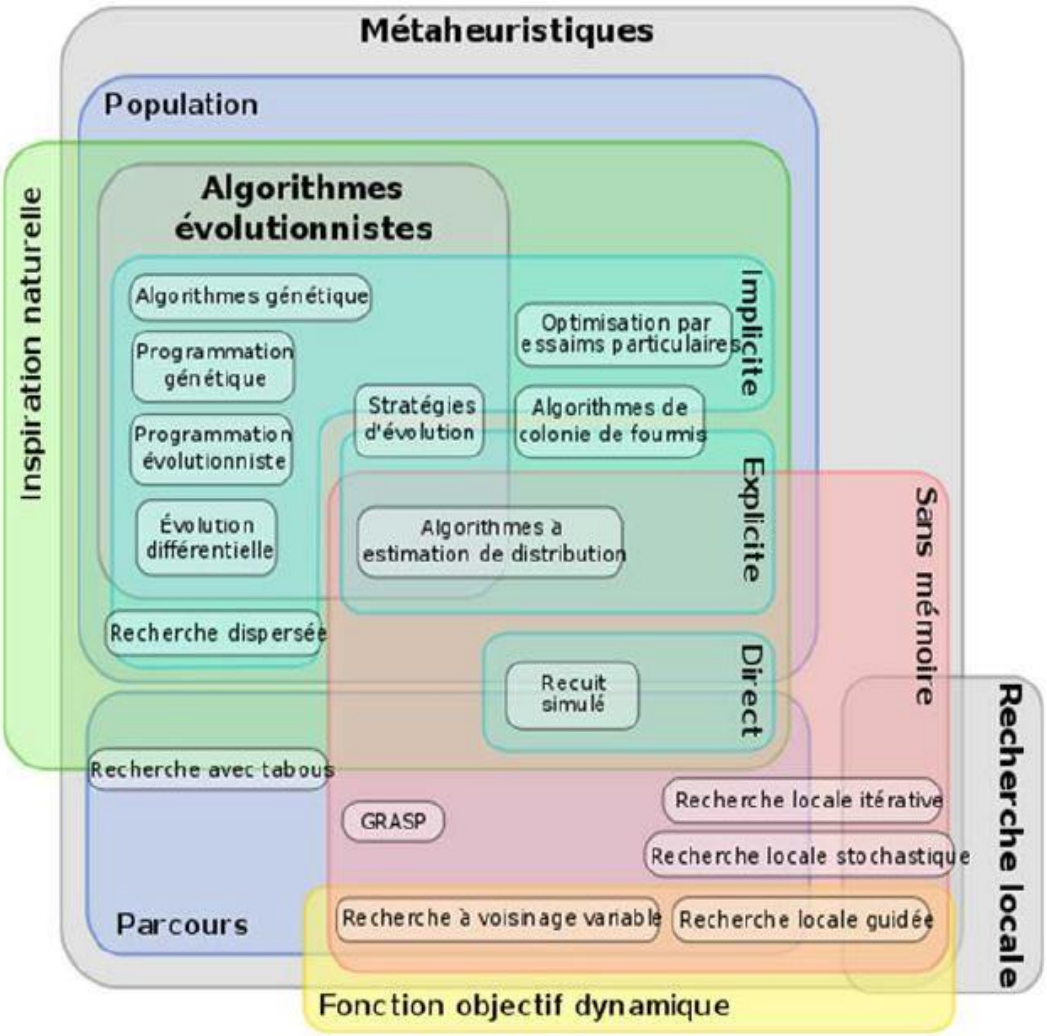


Figure 2.1 : Classification des métaheuristiques [19]

6. Principe des méthodes métaheuristique les plus répondues

6.1. Voisinage

Le principe général le plus largement utilisé dans l'élaboration des métaheuristiques est celui de voisinage. À chaque solution *s* du problème, on associe un sous ensemble *V*(*s*) de solutions. Ce sous-ensemble peut être statique, comme dans le cas du recuit simulé, mais aussi dynamique et dépend du temps *t* (ou plus précisément de l'itération à laquelle on se trouve).

Pour un problème d'optimisation combinatoire non convexe, pour lequel il est possible de définir un ensemble de voisinages a priori intéressants, la situation se complexifie. Il devient alors difficile de se décider pour l'un ou l'autre des voisinages autrement que par des essais. Comme le voisinage n'est qu'une partie des principes, tous interdépendants, utilisés dans l'heuristique, le choix d'un (ou de plusieurs) voisinage devient réellement problématique car on ne dispose que de très peu de résultats théoriques sur la qualité d'un voisinage pour un problème particulier.

Il existe une mesure de l'adéquation des voisinages appelée *rugosité* : l'idée consiste à évaluer la variance des coûts de deux solutions voisines. Cette variance est ensuite normalisée par la variance des coûts de l'ensemble des solutions et par la taille du problème, afin d'avoir une mesure indépendante de la valeur absolue de la fonction objectif ou de la taille du problème [20]

6.2. Recherche locale

La recherche locale, appelée aussi la descente ou l'amélioration itérative, représente une classe de méthodes heuristiques très anciennes.
Traditionnellement, la recherche locale constitue une arme redoutable pour attaquer des problèmes réputés très difficiles tels que le problème de voyageur de commerce et la partition de graphes. Contrairement à l'approche de construction, la recherche locale manipule des configurations complètes durant la recherche..[21]

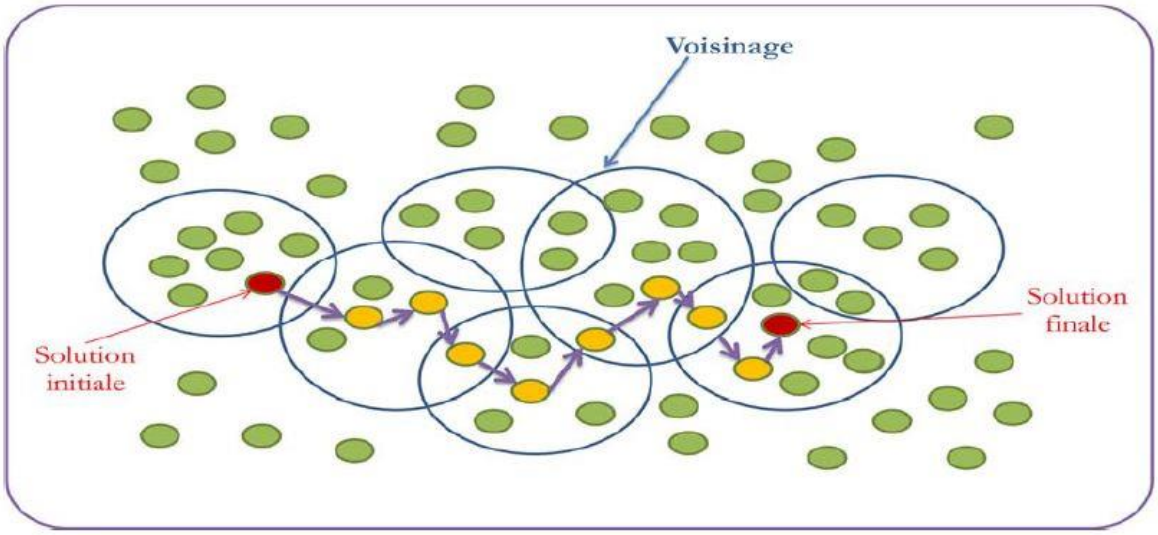


Figure 2.2 : Procédure générale de la recherche locale. [5]

6.2.1. Principe de recherche locale

Une méthode de recherche locale est un processus itératif fondé sur deux éléments essentiels : Un voisinage : $X \rightarrow 2x$et une procédure exploitant le voisinage. Plus précisément, elle consiste à débuter avec une configuration quelconque s de X, et choisir un voisin s'de s tel que $f(s') < f(s)$ et remplacer s par s'et à répéter jusqu'à ce que pour tout voisin s'de s, $f(s') \geq f(s)$.

Cette procédure fait intervenir à chaque itération le choix d'un voisin qui améliore la configuration courante. Plusieurs possibilités peuvent être envisagées pour effectuer ce choix. Il est possible d'énumérer les voisins jusqu'à ce qu'on en découvre un qui améliore strictement (première amélioration). Comme l'espace des solutions X est fini, cette procédure de descente s'arrête toujours, et la dernière configuration trouvée ne possède pas de voisin strictement meilleur qu'elle-même. Autrement dit, la recherche locale retourne toujours un optimum local. [21]

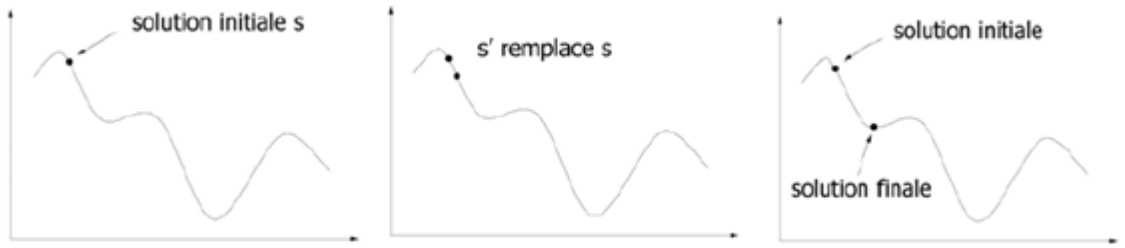


Figure 2.3 : Evolution d'une solution dans la méthode de recherche locale.[21]

6.2.2. L'algorithme PALS (Problem Aware Local Search)

Problem Aware Local Search est une variante du 2-opt de Lin qui est proposé par Luque et. Al dans [23].Cet heuristique n'utilise pas seulement le chevauchement entre les fragments, mais il tient également compte du nombre de contigs qui ont été créés ou détruits. C'est un algorithme très simple; fondamentalement, il consiste à évaluer à plusieurs reprises tous les mouvements possibles 2-Opt dans la solution actuelle en appliquant le meilleur.
La qualité d'un mouvement est évaluée en fonction de l'amélioration du nombre de contigs, ou du degré croissant de chevauchement des fragments dans le cas où aucune amélioration n'aurait pu être obtenue dans le nombre de contigs. [16]
L'une des principales clés de PALS est qu'il tient compte du nombre de contigs de la solution lors de l'évaluation de sa qualité. Étant donné que le calcul du nombre de contigs d'une solution est un processus coûteux, PALS l'estime en calculant le nombre de contigs créés ou supprimés lors de la manipulation de solutions provisoires. Contrairement à PALS, la plupart des techniques existantes dans la littérature ne considèrent pas le nombre de contigs (ce qui peut nous conduire à la situation indésirable dans laquelle une solution peut être meilleure que d'autres avec un plus grand nombre de contigs) ou calculer le nombre de contigs des solutions nouvellement générées avec une étape finale supplémentaire en utilisant un heuristique gourmand [16]. Le pseudo-code d'une méthode PALS est affiché dans l'algorithme 1.

Algorithme .1 : PALS

```
1  s ← generateInitialSolution() {créer une solution de départ}
2  répéter
3      L ← φ
4      pour i ← 1 à n − 1 faire
5          pour j ← i + 1 à n faire
6              ⟨Δf, Δc⟩ ← calculateDelta(s, i, j) {voir algorithme 2}
7              si (Δc < 0) ou (Δc = 0 et Δf > 0) alors
8                  L ← L ∪ ⟨i, j, Δf, Δc⟩ {Mémoriser tous les mouvements non
                       détériorants}
9              fin
10         fin
11     fin
12     si L ≠ φ alors
13         ⟨i, j, Δf, Δc⟩ ← selectMovement(L) {Sélectionner un mouvement}
14         applyMovement(s, ⟨i, j, Δf, Δc⟩) {Améliorer la solution}
15     fin
16 jusqu'à pas de changement
17 retourner s
```

L'algorithme commence évidement par la génération de la solution initiale qui sera éventuellement améliorée itérativement par trois étapes : Evaluer toutes les permutations possibles, Stocker dans une liste les permutations qui peuvent améliorer. Choisir et appliquer une permutation selon une stratégie bien précise.

Tout d'abord, pour générer la solution initiale, PALS met en œuvre deux méthodes. La première méthode génère un ordonnancement d'assemblage d'une façon complément aléatoire tandis que la seconde méthode emploie un algorithme glouton. Ensuite, une exploration du voisinage est faite en essayant toutes les permutations possibles en estimant à chaque fois les variables Δc et Δf engendrée par la permutation des deux fragments i et j.[22]

Les valeurs de Δc et Δf sont calculées par l'analyse du score de chevauchement entre (i , j, i-1 et j+1) et un seuil a qui détermine le score de chevauchement minimal. Δc et Δf sont utilisées pour décider l'adjacence de deux fragments. Le calcul de ces deux différences est présenté par la fonction CalculteDelta (algorithme 2). Une permutation candidate est stockée dans une liste de permutations, si elle est prometteuse pour réduire le nombre de contigs sinon elle sera rejetée.[22]

Algorithme .2 : calculateDelta(s, i, j) function

```
1  Δc ← 0
2  Δf ← 0
   {Calculer la variation du score de chevauchement :}
   {Ajouter le score de chevauchement des fragments affectés de la
    solution modifiée}
3  Δf ← w_{s[i−1]s[j]} + w_{s[i]s[j+1]}
   {Supprimer le score de chevauchement des fragments affectés de la
    solution courante}
4  Δf ← Δf − w_{s[i−1]s[i]} − w_{s[j]s[j+1]}
   {Calculer la variation du nombre de contigs :}
   {Incrémenter le nombre de contigs si un contig est coupé}
5  si w_{s[i−1]s[i]} > cutoff alors
6  |  Δc = Δc + 1
7  fin
8  si w_{s[j]s[j+1]} > cutoff alors
9  |  Δc = Δc + 1
10 fin
   {Decrémenter le nombre de contigs si deux contigs sont fusionnés}
11 si w_{s[i−1]s[j]} > cutoff alors
12 |  Δc = Δc − 1
13 fin
14 si w_{s[i]s[j+1]} > cutoff alors
15 |  Δc = Δc − 1
16 fin
17 retourner (Δf, Δc)
```

Une fois la liste établie et afin de choisir la permutation à appliquer, trois stratégie sont mises en œuvre : La meilleure permutation : elle consiste à choisir la meilleure solution qui améliore les deux métriques en favorisant celle qui réduit le plus le nombre de contigs. La première solution : elle a comme but de choisir la première permutation ajouté à la liste.

Aléatoire : une permutation est sélectionnée de façon aléatoire pour être appliquée. Cette méthode a donné des résultats encourageants en termes de temps et nombre de contigs mais le fait qu'elle utilise une méthode gloutonne dans la détermination des contigs, cela provoque une stagnation dans la qualité.[22]

7. Conclusion

Dans ce chapitre, nous avons présenté le principe de l'algorithme de recherche local PALS pour résoudre notre problème.

Le chapitre suivant sera consacré à la conception et l'implémentation de cet algorithme et la description des résultats obtenus.

Chapitre III :

Conception et implémentation

1. Introduction

Dans ce projet notre objectif est de proposer une version parallèle de l'algorithme de recherche local (PALS) nommé (PPALS) pour minimiser le temps de calcul.

Nous avons subdivisé notre chapitre en cinq sections, nous allons commencer par description du problème de travail, ensuite nous allons expliquer le cycle de projet, ensuite la description détaille de PALS et ces fonctions et la dernière section dédiées à la conception détaillée des deux versions (version parallèle et séquentielle).

2. Description du Problème

L'objectif de ce travail est d'implémenter la version séquentielle de l'algorithme PALS et de proposer une version parallèle pour améliorer le temps de calcul.

Figure 3.1 : Problématique du projet.

3. Cycle de Projet

Notre projet sera divisé en trois parties qui sont :

- ✓ Implémenter la version séquentielle de PALS.
- ✓ Paralléliser PALS et obtenir une version parallèle de (c'est-à-dire PPALS).
- ✓ Comparer les résultats des deux versions.

Toutes ces étapes sont décrites dans la figure.2

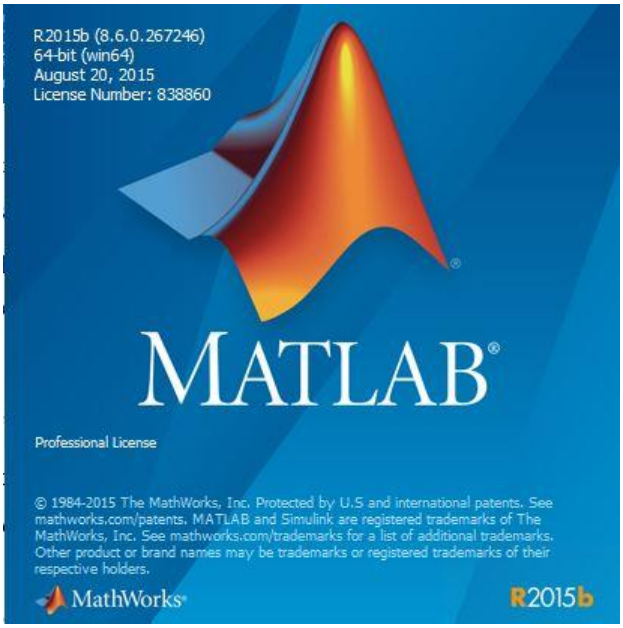
Figure 3.2 : Cycle de projet.

4. Environnement de développement

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

4.1. MATLAB

Est un langage de script émulé par un environnement de développement du même nom ; a été conçu par Cleve Moler à la fin des années 1970 ; il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran..



Les algorithmes étudies sont implémentes en utilisant le paradigme de logiciels et de matériel qui sont résumés dans le tableau suivant :

Matériel	Version
OS	Microsoft Windows 7 Professional, 64bits,
CPU	Intel(R) Core(TM) i3-3217U CPU @1.80GHz 1.80GHz
RAM	6.00Go

4.2. Base de données utilisée

Nous avons choisi différentes instances dans notre expérimental afin de tester et d'analyser la performance de notre algorithme

GenFrag prend une séquence d'ADN laosn et l'utilise comme brin parent à partir duquel des fragments aléatoires sont générés selon les critères fournis par l'utilisateur (longueur moyenne des fragments et couverture de la séquence parent). Nous avons choisi une grande séquence du site web du NCBI : une région ADN du CMH humain de classe II avec des répétitions de fibronectine de type II HUMMHCFIB, avec le numéro d'accession X60189 ; un apoloprotéme humain HUMAPOBF, avec le numéro d'accession M15421 ; le génome complet du

bactériophage lambda, avec le numéro d'accession J02459 ; et une séquence du BAC de Neurospora crassa, avec le numéro d'accession .38524243'.

Le tableau 3 présente des informations sur les ensembles de fragments spécifiques que nous utilisons pour tester notre algorithme.

Instance	Longueur de la séquence	Nom de l'instance	Nombre de fragments	Moyenne longueur Fragments	Couverture
X60189	3,835	X60189(4)	39	395	4
		X60189(5)	48	386	5
		X60189(6)	66	350	6
		X60189(7)	68	387	7
NM_007123	6,332	NM_007123(4)	37	691	4
		NM_007123(7)	64	680	7
M15421	10,089	M15421(5)	127	398	5
		M15421(6)	173	350	6
		M15421(7)	177	383	7
NC_001453	15,650	NC_001453(4)	90	708	4
		NC_001453(7)	157	676	7
NC_001807	16,571	NC_001807(4)	95	694	4
		NC_001807(7)	166	674	7
J02459	20,100	J02459(7)	352	405	7
BX842596	77,292	BX842596(4)	442	708	4
		BX842596(7)	773	703	7

Figure3.3: Tableau des instances de GenFrag

Nous avons utilisé deux fichier (fichier.dat,fichier.csv) :

-fichier.dat: présent la liste de fragments.

Les fragments utilisés sont représentés dans un tableau multidimensionnel contient deux champs (le numéro de fragment, contient le fragment lui-même) *Figure 4.*
-La commande matlab l'accès fréquent au fichier de fragments est :

```
% read fragments file
[filename, pathname,] = uigetfile('*.dat', 'Choisir le fichier de fragments');
name=[pathname filename];
[Header,Sequence]=fastaread(name)
```

listFrag

1x39 struct with 1 field

Fields	frag
1	'CACCATCTCAGGCCTGGAGCCAG...
2	'GGCCAGTAGGCAGTTGGTGGCCCT...
3	'TCTGGAGGGGCGCATTTGATGTAT...
4	'CATCTCAGGCCTGGAGCCAGACC...
5	'TTGCCTTCTGGAGGTTTCTAACAG...
6	'CCACCCAGCCCCAAGAATGGGCT...
7	'GAATTCCCAACCTCAGGTGATCCA...
8	'AGGGTAAAGATGCTCTGAGGCTGC...
9	'GCTGGTGAGGAGCTGGATCCAGG...
10	'ATGAAGCCGAGACCACCCAAGCA...
11	'TGGCCAGCGCGTGGGCCCCATCT...

Figure 3.4:Matrice de fragments.

-fichier.csv: présent la liste de chevauchement entre les fragments.

Les chevauchements entre les fragments sont représentés par Une matrice carrée de taille $n \times n$ *Figure 5.*

n: est le nombre de fragments.

-La commande matlab l'accès fréquent au fichier est :

```
%readoverlap file
 [filename, pathname,] = uigetfile('*.csv', 'Choisir le fichier chevach');
name=[pathname filename];
ftoread = name
fid = fopen(name, 'r')
```

chevauch

39x39 double

	1	2	3	4	5
1	0	11	48	304	10
2	11	0	23	11	20
3	48	23	0	48	225
4	304	11	48	0	13
5	10	20	225	13	0
6	12	11	193	27	107
7	13	9	8	13	13
8	11	331	23	11	11
9	10	302	23	10	10
10	163	9	255	160	48

Figure 3.5: Matrice de chevauchement

5. Description détaillée de PALS

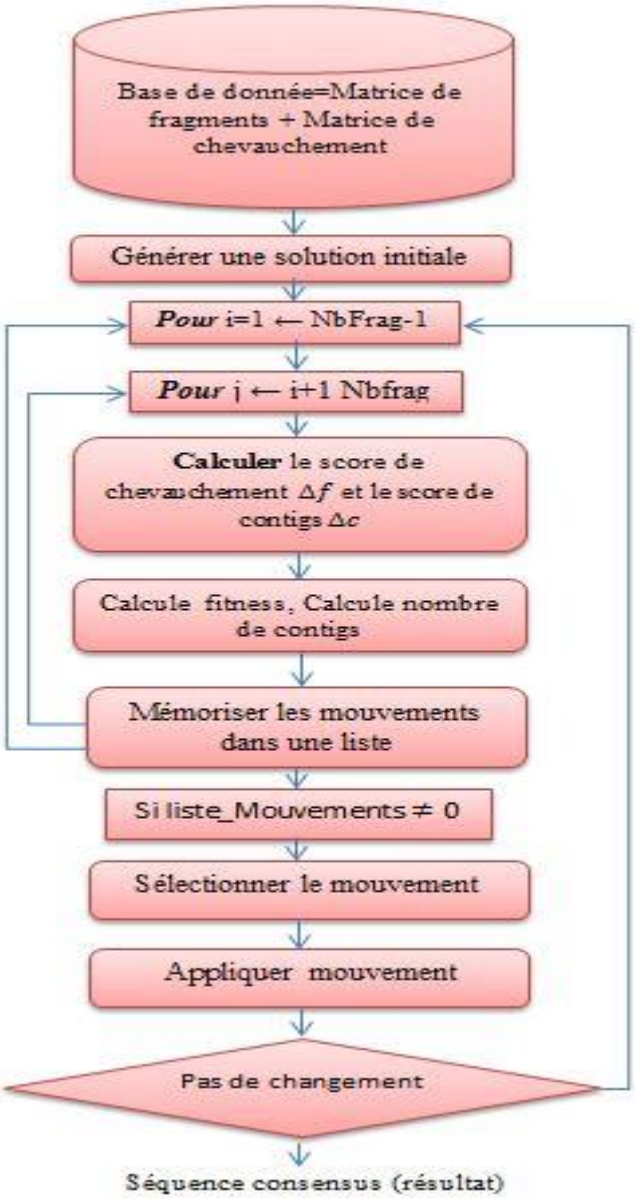


Figure3.6: l'algorithme PALS

5.1. Généré une solution initial

L'objectif de cette fonction est générer aléatoirement une solution initiale.
L'ensemble de fragments numérotés de 1 à Nbfrag, La solution de ce problème peut être représentée aléatoire des numéros de fragments pour trouver la séquence complète de génome.
Exemple: F =12 : Ensemble de 12 fragments. Alors (S=12)numérotés de 0 jusqu'à 11 ; *S*={0,1,2,3,4,5,6,7,8,9,10,11}
La solution est une permutation peut être un ensemble de combinaisons des éléments
-La commande matlab permant de générer aléatoirement un fragment de taille 'NbFrag (NbFrag : le nombre de fragments) est la suivante :

```
disp('..----- création dune solution initiale ----.....' );
solution = randperm(NbFrag);
```

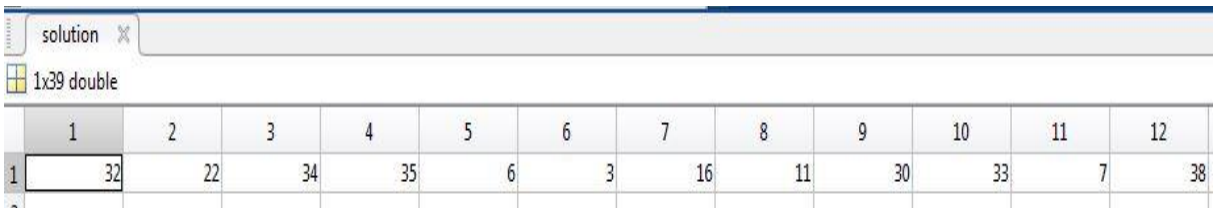


Figure 3.7: schéma illustratif d'une solution initiale

L'algorithme PALS démarre avec une solution initiale de n fragments générés de façon aléatoire.

5.2. Calculer le score de chevauchement Δf et le score de contigs Δc

L'objectif de la fonction Calcul_deltaF_deltaC permet d'est calculer le score de chevauchement Δf et le score de contigs Δc, il est possible de trouver une séquence avec le plus grande de score de chevauchements mais avec un nombre élevé de contigs.

Δc : Si un contig courant est coupé ou non après l'application d'un mouvement en se basant sur la valeur d'un paramètre *cutoff*.

Δf : Siune valeur de score calculé par la soustraction de la longueur de chevauchement des fragments affectés de la solution courante,

PALS utiliser les deux métrique $\Delta f, \Delta c$ pour évaluer la qualité des modifications

Algorithme 1: Calcul_deltaF_deltaC (S, chevauch, cutoff, NbFrag)
S : solution
Chevauch : matrice des chevauchements
NbFrag : nombre de fragments
Cutoff : seuil de chevauchement
Debut
Calculer deltaF
Calculer deltaC
Fin

Figure 3.8: Algorithme 1 Calcul_deltaF_deltaC

5.2.1. CalculateFitness

Dans cet algorithme Chaque individu de solution est évalué par la fonction fitness l'objectif de cette fonction est qui représente la performance de solution.

la fonction d'objectif (fitness) calcule la somme de chevauchement entre deux fragments de la solution *S*, de deux fragments successifs dans la solution.

$$F(s) = \sum_{i=1}^{n-1} w_{s[i],s[i+1]}$$

- *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].

```
Algorithme 2: Calculatefitness (solution,NbFrag,chevauch)

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

*Figure 3.9:*Algorithme 2: Calculatefitness

Exemple : la solution S=3 nombre de fragment f=9

Solution 1	1	5	0	3	2	6	4	7	8
Solution 2	8	6	1	5	7	0	3	4	2
Solution 3	2	4	1	5	6	0	7	3	8

Avec les fragments suivants :
F0 : CACAAGC
F1 : CTAGGAACTT
F2 : ATAC
F3 : GCACATATAC
F4 : GGAATTGA
F5 : TACTATCA
F6 : ACTAATTGGG
F7 : GATACAC
F8 : ACTAATT

La solution 1 :

1	5	0	3	2	6	4	7	8

F1 : CTAGGAACTT
F5 : TACTATCA
F0 : CACAAGC
F3 : GCACATATAC
F2 : ATAC
F6 : ACTAATTGGG
F4 : GGAATTGA
F7 : GATACAC
F8 : ACTAATT

($S1$)= 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+2=17

La solution 2 :

8	5	0	3	2	4	7	6	1

F8 : ACTAATT
F5 : TACTATCA
F0 : CACAAGC
F3 : GCACATATAC
F2 : ATAC----
F4 : GGAATTGA
F7 : GATACAC
F6 : ACTAATTGGG------
F1 : CTAGGAACTT

($S2$)= chevauchement(8,5)+ chevauchement(5,0)+ chevauchement (0,3)+ chevauchement(3,2) + chevauchement(4,7) + chevauchement(7,6)
=1+2+2+4+2+2=13

La solution 3 :

2	6	4	8	5	3	1	7	8

F2 : ATAC
F6 : ACTAATTGGG
F4 : GGAATTGA
F8 : ACTAATT
F5 : TACTATCA------
F3 : GCACATATAC
F1 : CTAGGAACTT------
F7 : GATACAC
F0 : CACAAGC

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

5.2.2. Calculer nombre de contigs

L'objectif de cette fonction est permet de calculer le nombre de contigs.

Algorithme 3 : Calcul_nombre_contigs (solution, chevauch, cutoff)

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

Figure 3.10: Algorithme 3Calcul_nombre_contigs

Exemple :
Soit la solution S1 : nContigs=1 parce que tous les fragments sont chevauchés.

1	5	0	3	2	6	4	7	8

F1 : CTAGGAACTT
F5 : TACTATCA
F0 : CACAAGC
F3 : GCACATATAC
F2 : ATAC
F6 : ACTAATTGGG
F4 : GGAATTGA
F7 : GATACAC
F8 : ACTAATT

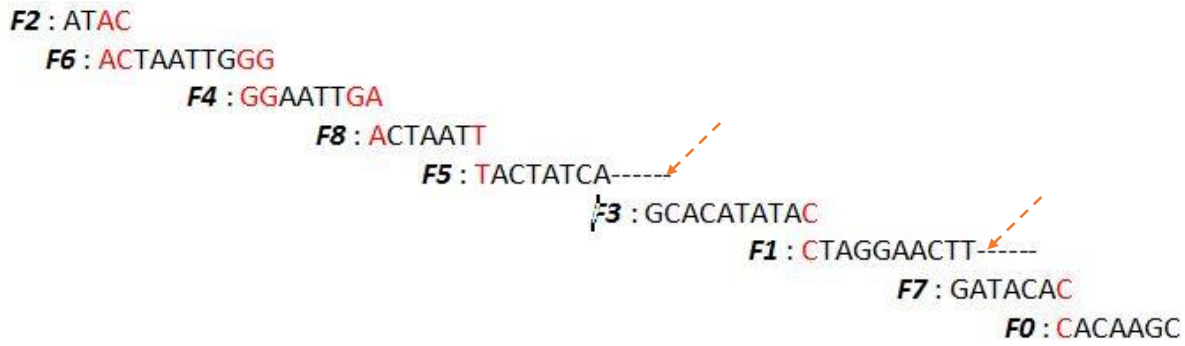
-Pas de coupure dans cette solution alors nContigs =1.
-fitness=17
Soit la solution S2 :

8	5	0	3	2	4	7	6	1

F8 : ACTAATT
F5 : TACTATCA
F0 : CACAAGC
F3 : GCACATATAC
F2 : ATAC----
F4 : GGAATTGA
F7 : GATACAC
F6 : ACTAATTGGG------
F1 : CTAGGAACTT

-Le nombre de contigs est : nContigs =3
fitness=13
Soit la solution S3 :

2	6	4	8	5	3	1	7	8



-Le nombre de contigs est : nContigs =3
-fitness=8

5.3. Mémoriser les mouvements dans une liste

L'objectif de cette fonction est de mémoriser les meilleurs mouvements dans une liste.

```
Algorithme 4: Liste_mouvement (i,j,Delta_F, Delta_C)
              S : solution

Debut
        Liste_mouvement(numMouv)← solution
        Liste_mouvement(i)← i
        Liste_mouvement(j) ← j
        Liste_mouvement(Delta_F)←Delta_F
        Liste_mouvement(Delta_C)←Delta_C
Fin
```

Figure 3.11:Algorithme 4 Liste_mouvement

5.4. Sélectionné mouvement

L'algorithme PALS permet d'améliore de façons itérative une solution initial, il utilise un ensemble de solution voisins N(s) pour générer la solution S, le mouvement consiste à inverser de i, j (positions de fragments dans la solution). Puis d'appliquer le mouvement ayant la plus petite valeur de$\Delta \boldsymbol{c}$. Dans le cas où il y a plusieurs mouvements avec la même valeur minimale de$\Delta \boldsymbol{c}$.on parfait le choix avec le mouvement ayant la plus grande valeur de$\Delta \boldsymbol{f}$. Plus formellement, un mouvement m donné par (i, j, $\Delta \boldsymbol{c}, \Delta \boldsymbol{f}$) est considère meilleur (en termes de degré d'amélioration à réaliser sur la solution courante) qu'un autre mouvement m' donné par (i', j', $\Delta \boldsymbol{c}', \Delta \boldsymbol{f}'$)si et seulement si les conditions suivantes sont vérifiées :

$$\Delta \boldsymbol{c} < \Delta \boldsymbol{c}' \textbf{ Ou } (\Delta \boldsymbol{c} = \Delta \boldsymbol{c}' \textbf{ et } \Delta \boldsymbol{f} > \Delta \boldsymbol{f}')$$

Algorithme 5 : Select_mouvement (liste_mouvement, solution)
Liste_mouvement : la liste des mouvements de la solution S
Debut
Calculatefitness (solution,NbFrag, chevauch)
Calcul_nombre_contigs (solution, chevauch, cutoff)
Selecte minimum nombre_contigs
sinombre_contigs(S) =nombre_contigs(Liste_mouvement) ***alors***
Select meilleure fitness
fin si
Retourne S
Fin

Figure 3.12:Algorithme 5 Select_mouvement

5.5. Appliquer mouvement

L'objectif de cette fonction est d'améliorer la solution courante c'est-à-dire remplacer la solution courante sur une tranche de solution sélectionné par la fonction select_mouvement.

5.6. Crétaire d'arrêt

Le critère d'arrêt est considéré par pas de changement c'est-à-dire pas de changent entre la solution courante et la solution modifiée.

6. Version parallèle de l'algorithme PALS(PPALS)

PPALS est un algorithme proposé dans ce travaille pour obtenir des résultats satisfaisants avec moins de temps d'exécution. Cet algorithme a les mêmes fonctions principales qu'PALS mais la seule différence est que PPALS attribue aux trois processus proc1, proc2 et proc3. Chaque processus appliquée même algorithme avec un parti de nombre de fragments

7. Description détaillée de PPALS

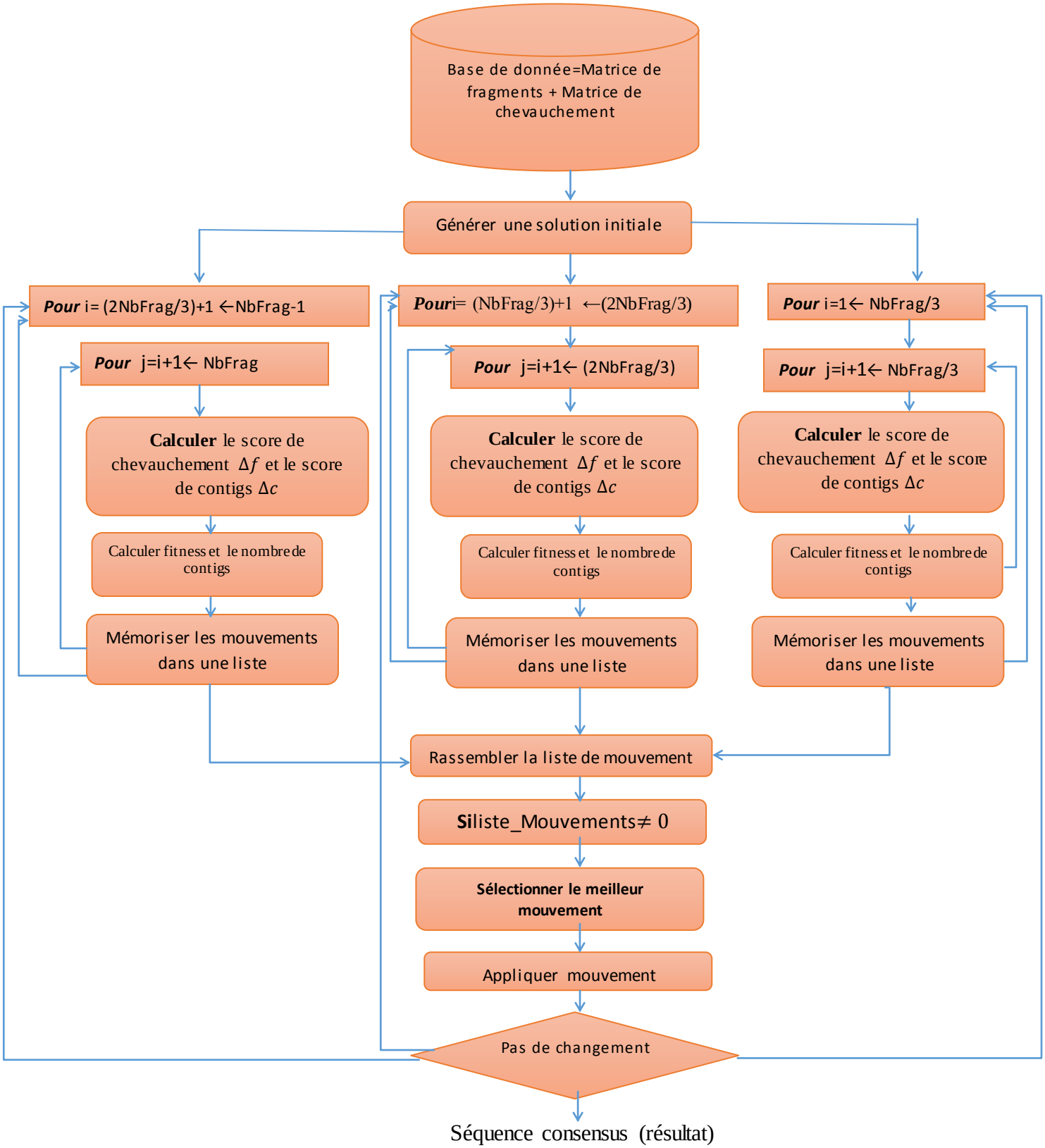


Figure 3.13:digramme de l'algorithme PPALS

8. implémentation et Résultats obtenus

8.1 importer le fichier.dat (fragments) et le fichier .csv (chevauchement)

```
% read fragments file
[filename, pathname,] = uigetfile('*.dat', 'Choisir le fichier de fragments');
 name=[pathname filename];
[Header,Sequence]=fastaread(name)

%read overlap file
[filename, pathname,] = uigetfile('*.csv', 'Choisir le fichier chevach');
name=[pathname filename];
ftoread = name
fid = fopen(name, 'r');

chevauch=csvread(name ,2,0); % 'chevauch' une matrice de chevauchement
%inserer les fragment dans la structure ListFrag
i=1;
  NbFrag = length(Sequence)
  while i <= NbFrag
      listFrag(i).frag= Sequence{1,i};
      i=i+1;
  end
```

Command Window

Figure 3.14: importer le fichier.dat (liste fragments) et le fichier .csv (chevauchement entre les frag)

9. Exemple

La matrice de séquence (fragments) utilisé sur l'exemple

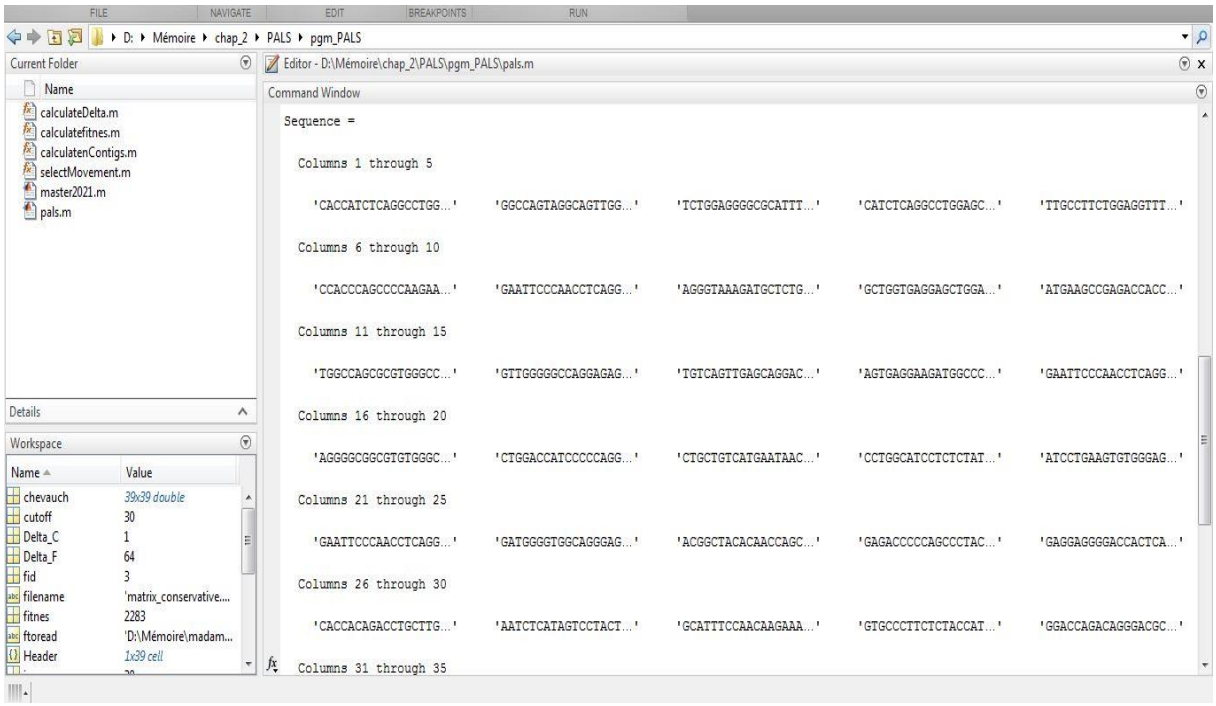


Figure 3.15: Liste des fragments

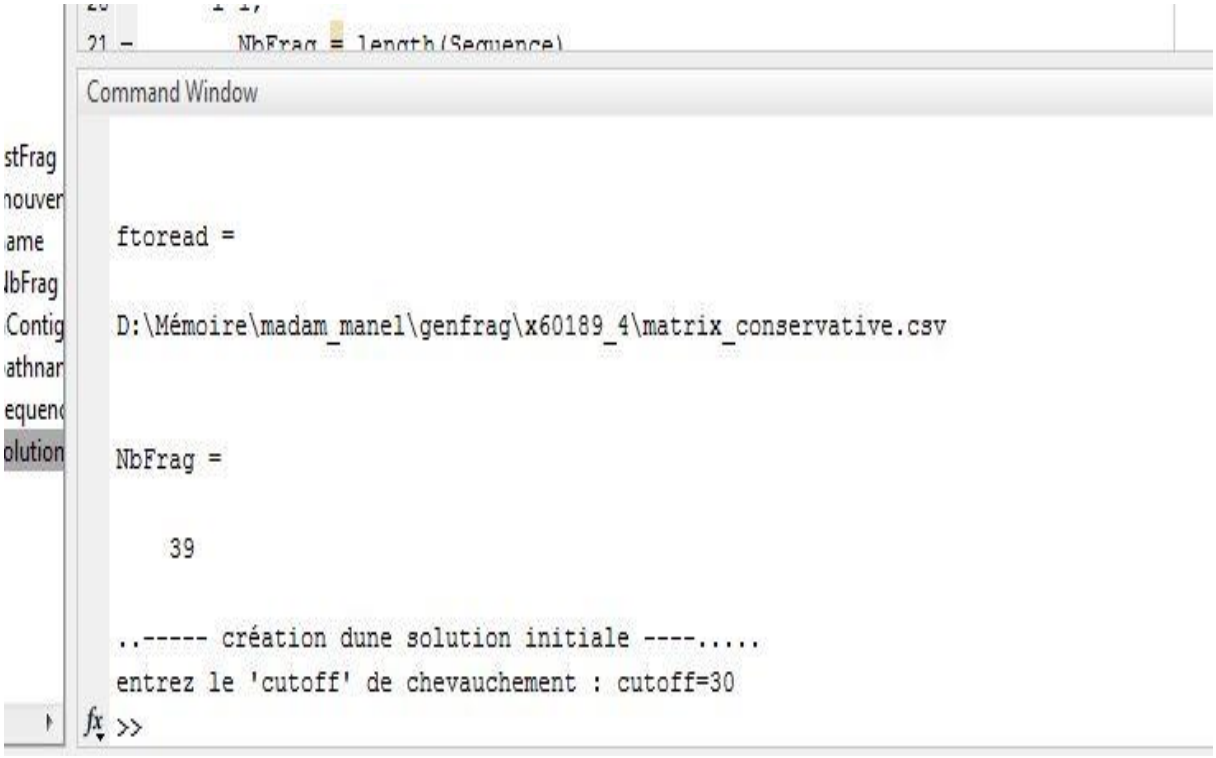


Figure 3.16: lire le cutoff

9.1. Les résultats

Figure 3.17: affichage de la solution.

Editor - pals.m | Variables - mouvement

mouvement

1x354 struct with 8 fields

Fields	numMouv	indMouv	i	j	Delta_F	Delta_C	nContigs	fitnes
178	178	1x39 double	12	22	-52	0	19	3851
179	179	1x39 double	12	25	5	0	19	3601
180	180	1x39 double	12	27	64	0	21	3264
181	181	1x39 double	12	28	-117	1	18	3931
182	182	1x39 double	12	29	62	0	19	3519
183	183	1x39 double	12	31	456	0	18	3608
184	184	1x39 double	13	14	235	2	18	3750
185	185	1x39 double	13	15	16	0	20	3495
186	186	1x39 double	13	16	355	1	19	4048
187	187	1x39 double	13	17	238	0	21	3543
188	188	1x39 double	13	18	231	0	21	3873
189	189	1x39 double	13	19	27	0	20	3787
190	190	1x39 double	13	21	300	1	21	3463
191	191	1x39 double	13	22	17	0	20	3663
192	192	1x39 double	13	23	250	0	20	3931

Figure 3.18: la liste de mouvement

Editor - pals.m | Variables - mouvement(10).indMouv

mouvement | mouvement(10).indMouv

mouvement(10).indMouv

	1	2	3	4	5	6	7	8	9	10	11	12
1	32	38	34	35	6	3	16	11	30	33	7	22
2												

Figure 3.19: afficher individué sur mouvement

Editor - pals.m | Variables - selectMouvement

mouvement | selectMouvement

1x39 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	24	15	7	8	9	14	12	31	38	5	1	29
2												

Figure 3.20: selectMouvement

Conclusion générale

Conclusion générale

Dans ce mémoire, nous avons utilisé l'algorithme de recherche local PALS pour résoudre le problème d'assemblage de fragments d'ADN. L'inconvénient majeur de cet algorithme est qu'il consomme beaucoup de temps d'exécution. Pour cette raison, PALS nécessitent une parallélisassions pour minimiser le temps d'exécution. Pour la satisfaction de l'utilisateur qui recherche toujours des résultats efficaces dans un temps réduit

Lors de la réalisation du projet, nous avons acquis des connaissances sur :

-Heuristique, métaheuristiques,

- Algorithmes de recherche local (pals)
- assemblage de fragments d'ADN

Nous avons implémenté cet algorithme en utilisant le langage de programmation Matlab et utilisé l'instance d'ADN généré par l'outil GenFrag.

Bibliographie

[1]- Ali, A. B., Luque, G., Alba, E., &Melkemi, K. E. (2017). An improved problem aware local search algorithm for the DNA fragment assembly problem.*Soft Computing*, *21*(7), 1709-1720.

[2]- NILGES, Michael et LINGE, Jens P. Bioinformatics-a definition. *Unité de Bio–Informatique Structurale, Institut Pasteur*, 2011, p. 25-28.

[3]- Abdesslem, L. (2017). Approche quantique évolutionnaire pour l'alignement multiple de séquences en bioinformatique

[4]- Taing, L. (2012). *Approches bioinformatiques pour l'exploitation des données génomiques* (Doctoral dissertation).

[5]- BOUHALI R, DJERROUD Y.(2013). Algorithme de recherche locale pour le problème d'assemblage de fragments d'ADN sur processeurs graphiques (GPU)

[6]-(21/01/2021).site :http://biochimej.univ-angers.fr/Page2/COURS/9ModulGenFoncVeg/5MethEtudGenFonc/1MethodeSEQUENGAGE/1SEQUENGAGE.htm

[7]-Lamoril, J., Ameziane, N., Deybach, J. C., Bouizegarene, P., &Bogard, M. (2008). Les techniques de séquençage de l'ADN: une révolution en marche. Première partie. *Immuno-analyse & Biologie Spécialisée*, *23*(5), 260-279

[8]- Théroux, J. F. (2015). Développement de méthodes d'assemblage de génomes de novo adaptées aux bactéries endosymbiotes.

[9]- HUANG, Ko-Wei, CHEN, Jui-Le, YANG, Chu-Sing, *et al.*A memeticparticleswarmoptimizationalgorithm for solving the DNA fragment assemblyproblem. *Neural Computing and Applications*, 2015, vol. 26, no 3, p. 495-506.

[10]-BRUTO, Maxime. *Étude de la plasticité génomique chez Streptomycesambofaciens: assemblage et analyse comparative du génome des souches ATCC23877 et DSM40697*. 2010. Thèse de doctorat. UHP-Université Henri Poincaré.

[11]-Yolanda Smith,B.Pharm,(20/01/2021). Assemblée de séquence d'ADN (https://www.news-medical.net/life-sciences/DNA-Sequence-Assembly-(French).aspx)

[12]- HAO, Jin-Kao, GALINIER, Philippe, et HABIB, Michel. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'intelligence artificielle*, 1999, vol. 13, no 2, p. 283-324.

[13]- BEN ALI, A. (2018). *Contributionsa la r´ esolution de problemes d'optimisation combinatoires NP-difficiles* (Doctoral dissertation, UNIVERSITE Mohamed Khider Biskra).

[14]- Lucien MOUSIN 2018,Doctora,Extraire et exploiter la connaissance pour mieux optimisZ

[15]- Drouiche, A., Cheniguel, L., &Ghanem, S. (2020). *Algorithmes Bio-inspirés pour L'optimisation du Routage dans les Réseaux Ad Hoc* (Doctoral dissertation, Univ. A/Mira-Bejaia).

[16]- Ghrib Ramadhan,(2014) , A HybridMetaheuristicAlgorithm for Solving the DNA Fragment AssemblyProblem(Mémoire de Master Université d‟El-Oued).

[17]- YahyaBouzoubaa. Méthodes exactes et heuristiques pour l'optimisation de l'agencement d'un logement : application aux situations de handicap. Modélisation et simulation. Université de Lorraine, 2017. Français.

[18]-https://waytolearnx.com/2019/03/difference-entre-un-probleme-np-complet-et-np-difficile.html 32/05/2021

[19]https://fr.wikipedia.org/wiki/M%C3%A9taheuristique#/media/Fichier:Metaheuristics_classification_fr.svg (18-05-2021)

[20]- Slimani, L. (2009). *Contribution à l'application de l'optimisation par des méthodes métaheuristiques à l'écoulement de puissance optimal dans un environnement de l'électricité dérégulé* (Doctoral dissertation, Université de Batna 2).

[21]-L. Kherbouche, Z. Oubahri, 2017, Quelques méthodes de résolutions en optimisation combinatoire,(MEMOIRE DE MASTER)]

[22]-ZEMALI El-amine,(2017) Contribution à la Résolution Coopérative Approchée De Problèmes en Bioinformatique : Assemblage de Fragments d'ADN et Alignement de Séquences Multiple (Diplôme de Doctorat USTHB)

[23]-Enrique Alba, Gabriel Luque et Sami Khuri.(2005)Assembling DNA fragments withparallelalgorithms. Dans Evolutionary Computation,