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 : SIOD01/M2/2022

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : Systèmes d'information, Optimisation et Décision (SIOD)

Traitement parallèle pour l'assemblage de fragments d'ADN

Par :

CHIBA ABDALLAH

Soutenu le 21/06/2023 devant le jury composé de :

GUERROUF Fayçal	MCB	Président
Moussaoui Manel	MAA	Rapporteur
Chighoub Fouzia	MAA	Examinateur

Année universitaire 2022-2023

Remerciements

*Tout d'abord, je tiens à remercier **ALLAH**, le Tout-Puissant, de m'avoir donné le courage et la patience nécessaires pour terminer ce modeste travail.*

*Je tiens à exprimer ma profonde gratitude envers mon encadreur, **Mme Moussaoui Manel**. Votre patience, vos conseils et votre soutien tout au long de ce parcours ont été inestimables. Votre expertise et votre passion m'ont inspiré et motivé à donner le meilleur de moi-même. Merci pour votre confiance et votre guidance précieuse.*

Je souhaite également remercier chaleureusement l'ensemble de l'équipe pédagogique du département d'informatique de notre institution, pour leur engagement et leur dévouement à fournir un enseignement de qualité. Leurs connaissances approfondies et leur enthousiasme ont grandement contribué à ma formation et à ma compréhension des concepts clés dans le domaine de l'informatique.

Enfin, mes remerciements vont également à mes amis et à ma famille pour leur soutien inconditionnel et leurs encouragements tout au long de mes études. Leurs mots d'encouragement et leur présence ont été des sources de motivation essentielles dans les moments difficiles.

Dédicace

Je dédie ce modeste travail à :

La mémoire de mon père.

ma mère.

À mes frères et sœurs, Safiya, Asmaa et Abdelrahman.

À toute ma famille et à 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. L'algorithme de recherche local (PALS) offre une approche itérative pour trouver une solution optimale ou quasi optimale à ce problème. Cependant, le principal inconvénient de PALS réside dans son temps d'exécution croissant avec la taille de la base des fragments. Pour résoudre ce problème, nous proposons une version parallèle de PALS appelée PPALS, qui permet d'obtenir des résultats efficaces avec un temps d'exécution considérablement réduit. Les résultats expérimentaux démontrent que PPALS améliore significativement PALS en minimisant le temps de calcul.

Mots-clés : assemblage de fragments d'ADN, PALS, PPALS, Contigs.

Abstract

The fragment assembly problem is that of reconstructing a DNA sequence from several hundred (or even thousands) of fragments obtained by biologists in the laboratory. The Problème Aware Local Search (PALS) algorithm offers an iterative approach to find optimal or near-optimal solution. However, the main drawback of PALS lies in its increasing execution time with the size of the fragment benchmark. To solve this problem, we propose a parallel version of PALS called PPALS, which provides efficient results with significantly reduced execution time. Experimental results demonstrate that PPALS significantly improves upon PALS by minimizing the computation time.

Keywords : DNA fragment assembly, PALS, PPALS, Contigs.

ملخص:

تركز هذه الأطروحة على مشكلة تجميع شظايا الحمض النووي ، وتسلط الضوء على خوارزمية البحث المحلي (PALS) التي تعتبر من ابرز الخوارزميات التي تعالج هذا المشكلة. ولكنها تواجه مشكلة زيادة وقت التنفيذ مع زيادة حجم قاعدة بيانات الشظايا. تهدف هذه الدراسة للاستفادة من تقنيات التوازي لتحسين وقت التنفيذ من خلال انشاء خوارزمية PPALS النسخة الموازية لخوارزمية PALS لتسريع عملية تجميع شظايا الحمض النووي ، يهدف PPALS إلى تقليل وقت التنفيذ مع الحفاظ على نتائج فعّالة. تؤكد النتائج المتحصل عليها إمكانية تحسين الوقت اللازم لعملية تجميع شظايا الـ DNA باستخدام تقنيات المعالجة المتوازية، مما يفتح الباب أمام البحوث المستقبلية في هذا المجال.

الكلمات المفتاحية : تجميع شظايا الحمض النووية , Contigs , PPALS , PALS

Sommaire

Table des figures

Liste des tableaux

Introduction générale

Le problème d'assemblage de fragments d'ADN est un défi majeur dans le domaine de la bioinformatique. Il consiste à trouver un ordre total des fragments d'ADN donnés, permettant de reconstruire avec précision la séquence parente. Le séquençage de l'ADN est devenu une pratique essentielle dans divers domaines tels que la médecine, la phylogénétique et la criminalistique. Il implique la génération de séquences d'ADN à partir de centaines voire de milliers de fragments obtenus en laboratoire.

Ce problème complexe a émergé dans le domaine de la bioinformatique pour automatiser le séquençage de grandes séquences d'ADN. Il est classé comme un problème NP-difficile, avec un nombre exponentiel de configurations possibles pour n fragments.[14]

L'algorithme de recherche locale PALS (Problème Aware Local Search) a été développé par Alba et Luque en 2007 pour résoudre le problème d'assemblage de fragments d'ADN .Dans la littérature PALS est l'une des methode heuristiques les plus efficaces pour ce problème et il donne d'assez bonnes solutions [14]

Cependant, l'algorithme PALS rencontre des limitations de performance lorsqu'il est confronté à des instances de grande taille du problème d'assemblage de fragments d'ADN. Avec une augmentation du nombre de fragments, l'algorithme peut prendre plus de temps pour trouver une séquence consensus précise, ce qui limite son utilité pratique. Cela souligne la nécessité d'améliorer les performances de l'algorithme PALS pour résoudre efficacement les instances de grande taille.

Dans ce travail une version parallèle de PALS est proposée pour obtenir des bons résultats avec un temps d'exécution bien inférieur à celui de la version séquentielle sans perte de précision significative

Ce mémoire débute par une introduction générale qui présente la problématique et l'objectif du projet.

Il se compose de trois chapitres : Le premier chapitre est composé de deux parties : la première partie décrit une introduction au problème d'assemblage de fragments d'ADN et la deuxième partie explique

les méthodes métaheuristiques les plus répandues, en mettant l'accent sur le recherche locales. Ensuite, il décrit l'algorithme PALS en détail

Le deuxième chapitre montre les concepts de base du calcul parallèle organises en trois sections principales : les concepts lies à la parallélisassions, les modèles de programmation parallèle et les type de parallélisme

Le troisième chapitre se concentre sur la conception , l'implémentation et la parallélisassions de PALS avec la démonstration de son efficacité. Ce chapitre est composé de 2 partie. Partie 01 : décrit la conception globale et détaillé de notre système (conception de PPALS) et Partie 02 : illustre l'implémentation et la comparaison entre les deux algorithmes (c.-à-d. PALS, PPALS).

Le mémoire se termine par une conclusion générale qui évalue les résultats obtenus et discute quelques perspectives.

Chapitre 1

Problème d'assemblage de fragments d'ADN et Algorithme PALS

1.1 Introduction

L'assemblage de fragments d'ADN est le défi de reconstruire une séquence d'ADN à partir de fragments obtenus par des biologistes de laboratoire. C'est un sujet important dans la bio-informatique pour les projets de génome.Ce chapitre est divisé en deux parties. Dans la première partie, 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. Dans la deuxième partie aborde les approches métaphysiques les plus couramment utilisées, en se concentrant spécifiquement sur les recherches locales. en suite,l'algorithme PALS sera présenté en détail.

1.2 Partie 1 : Problème d'assemblage de fragments d'ADN

Dans cette partie, nous allons présenter une étude détaillée sur le problème d'assemblage de fragments d'ADN tel que : les notions de base,les étapes d'analyse d'ADN puis nous décrirons le processus de séquençage ADN et enfin nous exposons les différentes méthodes et les techniques d'assemblage.

1.2.1 Bioinformatique

La collecte de données biologiques s'est considérablement développée ces dernières années, grâce notamment au développement de nouvelles méthodes de compréhension de l'ADN et d'autres composants du vivant. Pour analyser ces données plus volumineuses et plus complexes, les scientifiques se sont tournés vers les nouvelles technologies de l'information. L'énorme capacité des ordinateurs à stocker et à analyser des données leur permet de mener des recherches. L'intersection de la biologie et de l'informatique s'appelle la bio-informatique. Cela couvre les disciplines des sciences de la vie telles que la génomique, la protéomique et la biologie des systèmes.

1.2.1.1 Qu'est-ce que la bio-informatique ?

La bioinformatique est la discipline qui consiste à analyser, comprendre et utiliser des données biologiques à l'aide d'outils informatiques. Il s'applique à de nombreux domaines tels que la génomique, la protéomique, la métabolomique et la recherche sur les réseaux biologiques.Par exemple les algorithmes mathématiques sont utilisés pour traiter les grandes quantités de données produites par les technologies de séquençage modernes, telles que le séquençage Shotgun, afin de générer des informations sur la structure et la fonction des génomes, des protéines et des réseaux biologiques. La bioinformatique est un domaine en pleine croissance et est considérée comme l'une des disciplines les plus importantes pour comprendre et relever les défis de la médecine et de la biologie au 21e siècle [15].

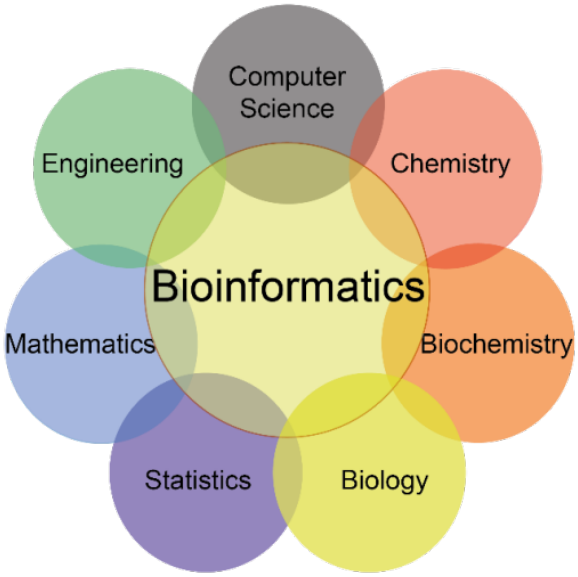

FIGURE 1.1 – Les sciences qui constituent la bio-informatique [1]

1.2.1.2 Champs d'applications de la bioinformatique

Des domaines d'application divers et variés se sont établis au sein de la bioinformatique, formant ainsi de nombreuses sous-disciplines :

- La bioinformatique des séquences, analyse les données génétiques dans les séquences d'ADN et de protéines. Cette sous-discipline se concentre sur la reconnaissance des similitudes entre les séquences, la découverte de gènes ou de régions biologiques pertinents dans l'ADN ou les protéines en utilisant la succession des éléments fondamentaux (nucléotides, acides aminés).[16]
- La bioinformatique structurale s'occupe de la modélisation en 3D et du repliement des macromolécules biologiques (protéines et acides nucléiques) en utilisant des algorithmes informatiques.[16]
- La bioinformatique des réseaux étudie les relations entre les gènes, protéines, cellules, organismes en analysant et modélisant les comportements collectifs de ces éléments de la vie. Cette discipline utilise principalement les données de technologies de haut débit telles que la protéomique et la transcriptomique pour comprendre les échanges génétiques et métaboliques.[16]

1.2.2 Analyse de l'AND

1.2.2.1 Définition d'ADN

L'ADN (acide désoxyribonucléique) est la molécule biologique qui stocke les informations génétiques des êtres vivants. Elle est formée par une longue chaîne d'unités appelées nucléotides, qui sont les briques de base de l'ADN. Les nucléotides sont reliés les uns aux autres par des liaisons phosphodiestères et sont formés de sucre désoxyribose, de phosphate et de bases nitrogenées (adenine, guanine, cytosine et thymine). L'ordre précis des nucléotides dans l'ADN détermine les informations géné-

tiques spécifiques d'un organisme. Les informations génétiques sont codées dans la séquence d'ADN et peuvent être transmises d'une génération à l'autre [17].

1.2.2.2 Structure de l'ADN

La structure de l'ADN est en double hélice, formée par deux brins en spirale qui s'enroulent autour d'un axe central. Chaque brin est composé de nucléotides, les unités de base de l'ADN. Les nucléotides comprennent un groupement phosphates, un sucre désoxyribose et une base nitrogenée (adénine, guanine, cytosine ou thymine). Les deux brins de l'ADN sont tenus ensemble par des ponts de liaison hydrogène entre les bases nitrogenées. Les bases nitrogenées sont associées en paires spécifiques : adénine avec thymine, et guanine avec cytosine. Cette association en paires de base est appelée la "complémentarité des bases". La figure 1.2 décrit la structure de l'AND .[17].

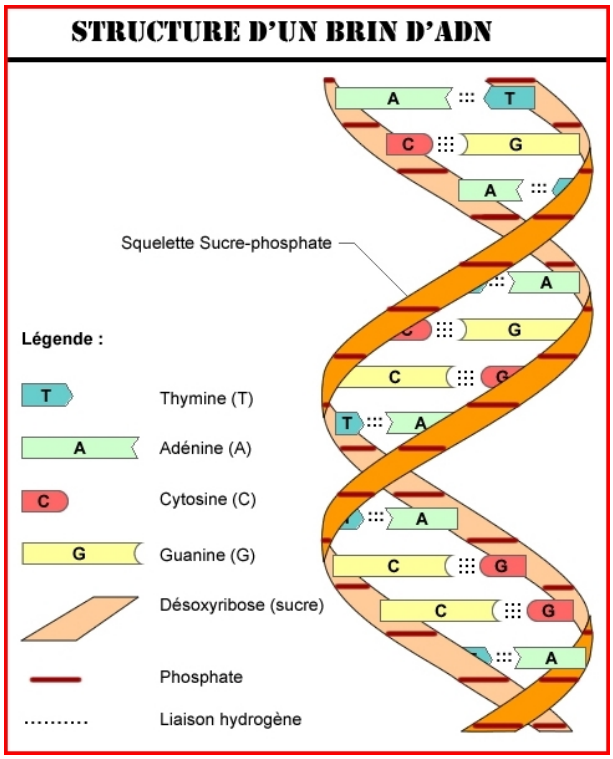

FIGURE 1.2 – Structure de l'AND [2]

1.2.2.3 Les étapes d'analyse de l'ADN

L'obtention de l'ADN d'un organisme étudié peut être effectuée à partir de l'ensemble ou d'une partie de celui-ci. L'extraction de l'ADN peut impliquer l'utilisation d'un grand nombre de cellules, mais la quantité finale d'ADN obtenue est souvent limitée. Pour surmonter cette limitation, il est nécessaire d'amplifier cette quantité d'ADN afin de pouvoir l'analyser. La séquence de l'ADN est très longue, avec un génome humain de taille approximative de 3,2 milliards de paires de nucléotides, il est donc nécessaire de la diviser en fragments plus petits pour l'étude. la figure 1.3 représenté processus d'analyse de l'ADN[18].

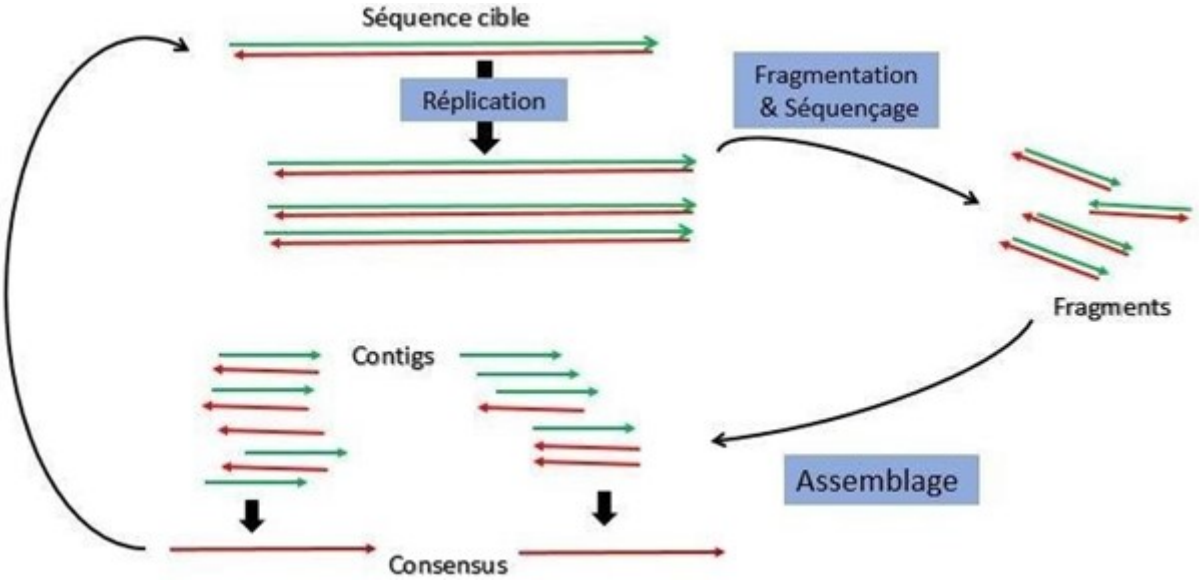

FIGURE 1.3 – Processusd'analysed'ADN[3]

1.2.2.3.1 Réplication de la séquence d'ADN

La réplication de la séquence d'ADN est le processus de duplication ou d'amplification d'un segment spécifique d'ADN. Cela se fait à l'aide de la technique de PCR (Polymerase Chain Reaction), qui utilise l'enzyme ADN polymérase pour dupliquer plusieurs millions de fois le fragment d'ADN cible. La PCR est un outil crucial dans de nombreuses applications biologiques et médicales, telles que la détection de maladies génétiques, l'identification de gènes, et la cartographie génétique [10]. La duplication de l'ADN est une mutation qui se caractérise par la multiplication d'une partie de la séquence génomique. Elle peut couvrir tout le génome, un chromosome entier ou un fragment de chromosome plus ou moins important. Cela peut aboutir à la formation de copies multiples d'un ou plusieurs gènes, ce qui entraîne une redondance de l'information génétique. Les duplications peuvent être à l'origine de certaines maladies génétiques et peuvent également jouer un rôle important dans l'évolution de l'espèce. Le principe de replication est illustrée dans la figure 1.4 [6].

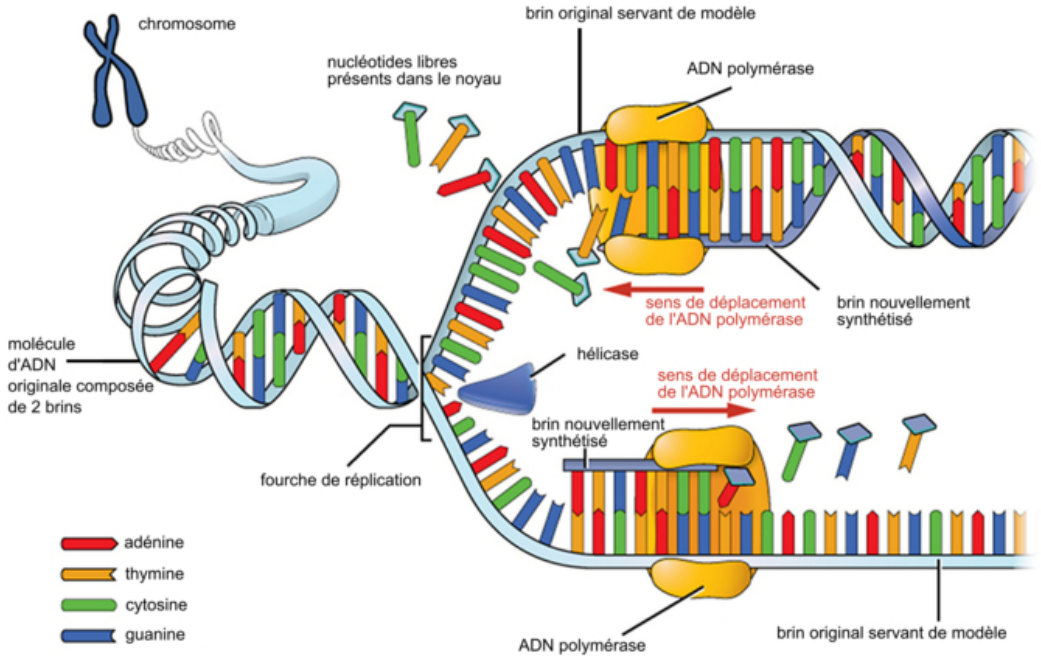

FIGURE 1.4 – Duplication de l'ADN [4]

1.2.2.3.2 Fragmentation de l'ADN

L'analyse d'une molécule d'ADN de grande taille implique la division en fragments de tailles appropriées pour le séquençage. Deux méthodes principales de fragmentation ont été largement utilisées pour atteindre ce but :

- **La fragmentation aléatoire :** On divise directement l'ensemble de l'ADN à séquencer en de petits bouts de taille optimisée pour le séquençage (généralement 1000 paires de bases)[4].
- **Segmentation après cartographie :** Dans les cas de génomes très volumineux, la tâche de reconstruction peut s'avérer compliquée. Par conséquent, certaines équipes ont adopté une approche en deux étapes appelée "segmentation après cartographie" pour faire face à ce défi[3].

1.2.2.3.3 L'assemblage de fragments d'ADN

L'assemblage de l'ADN est l'étape au cours de laquelle les fragments d'ADN sont réunis pour reconstituer la séquence complète de l'ADN d'origine. Cette étape revêt une importance cruciale car elle permet de reconstruire la séquence d'origine à partir de laquelle ces fragments ont été générés

1.2.3 Séquençage de fragments d'ADN

L'étape initiale pour l'analyse du génome d'un organisme donné consiste à déterminer la séquence complète de l'ADN, qui est le processus de séquençage de l'ADN. Les informations fournies par ce

processus sont utilisées pour identifier les fonctions des gènes dans le génome.

Le séquençage de l'ADN implique la détermination de la séquence des nucléotides d'un fragment d'ADN particulier. Cette tâche est accomplie grâce à des appareils de séquençage appelés séquenceur [6].

Un appareil nommé séquenceur de gènes peut automatiser la tâche de séquencer l'ADN. Il détermine l'ordre des bases A, C, T et G en allant de gauche à droite sur une hélice double d'ADN, jusqu'à une longueur standard, avec un taux d'erreur commun. La séquence courte fournie par le séquenceur est appelée une "lecture" d'après le terme anglais "read". La figure (1.5) montre un exemple de séquenceur d'Illumina. Les capacités d'un séquenceur peuvent être déterminées par :

- La longueur des lectures produites (L).
- Le nombre total de lectures produites (n).
- Le nombre total de nucléotides lus (L x n).
- Le temps nécessaire pour le séquençage.
- La qualité du résultat du séquençage.

FIGURE 1.5 – Un séquenceur d'ADN (Illumina) [1]

1.2.3.1 Les méthodes de séquençage

Suite à l'arrivée de la technique de séquençage de l'ADN, de nombreuses méthodes de séquençage ont été élaborées :

1.2.3.1.1 Séquençage de première génération

Le séquençage de l'ADN a été découvert au milieu des années 1970. En 1977, deux méthodes ont été inventées séparément, l'une par l'équipe de Maxam et Gilbert aux États-Unis et l'autre par Frederick

Sanger au Royaume-Uni. Ces deux méthodes sont basées sur des principes différents : la méthode de Sanger utilise une synthèse enzymatique sélective, tandis que celle de Maxam et Gilbert utilise une dégradation chimique sélective.

À la fin des années 1980, la méthode de Sanger a connu une automatisation grâce à l'utilisation de marquages fluorescents et d'électrophorèse capillaire. Cela a permis de se diriger vers le séquençage à haut débit en simplifiant et en accélérant le processus. La technique de séquençage à l'aide de didésoxyribonucléotides fluorescents (ddNTP) utilise des didésoxyribonucléotides marqués chacun par un fluorophore distinct. Les fragments d'ADN synthétisés portent ce fluorophore terminal et sont appelés "terminateurs d'élongation" ou "Big DyeTerminators" ou encore "Dye-labeledterminator"[19].

1.2.3.1.2 Séquençage de deuxième génération (NSG)

Le terme NGS (Next GenerationSequencing) désigne les différentes technologies de séquençage développées depuis 2005 par des entreprises de biotechnologie. Cette nouvelle technique est considérée comme la seconde génération de séquençage. Avec ces nouvelles méthodes, il est possible de séquencer de grandes quantités d'ADN en très peu de temps, et la taille des fragments peut varier de 150 à 300 pb [20] :

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

Les progrès technologiques ont considérablement transformé le secteur du séquençage de l'ADN, avec l'arrivée des technologies de séquençage NGST à haut débit (next-génération high throughput DNA sequencing technologies) depuis 2012, une véritable révolution en génomique fonctionnelle a eu lieu. Autrefois, séquencer 800 à 1000 nucléotides pouvait prendre plusieurs jours avec des méthodes complexes, lourdes et potentiellement dangereuses impliquant l'utilisation d'isotopes radioactifs. Aujourd'hui, les techniques de séquençage sont plus simples et peuvent générer des millions de nucléotides en un jour. Les données de séquençage sont immédiatement intégrées dans des bases de données pour une analyse en temps réel.

1.2.3.2 Le séquençage de Shotgun

Le séquençage Shotgun est une méthode de séquençage génétique qui a été introduite en 1982 par Frederick Sanger. Elle consiste à découper de manière aléatoire le génome en sections prédéterminées de longueurs telles que 2 000, 10 000 ou 50 000 paires de base. Après séquençage de chaque fragment, des algorithmes mathématiques sont utilisés pour assembler les fragments adjacents et déterminer leur position exacte sur le génome.La figure 1.6 montre les étapes de Séquençage de Shotgun [21].

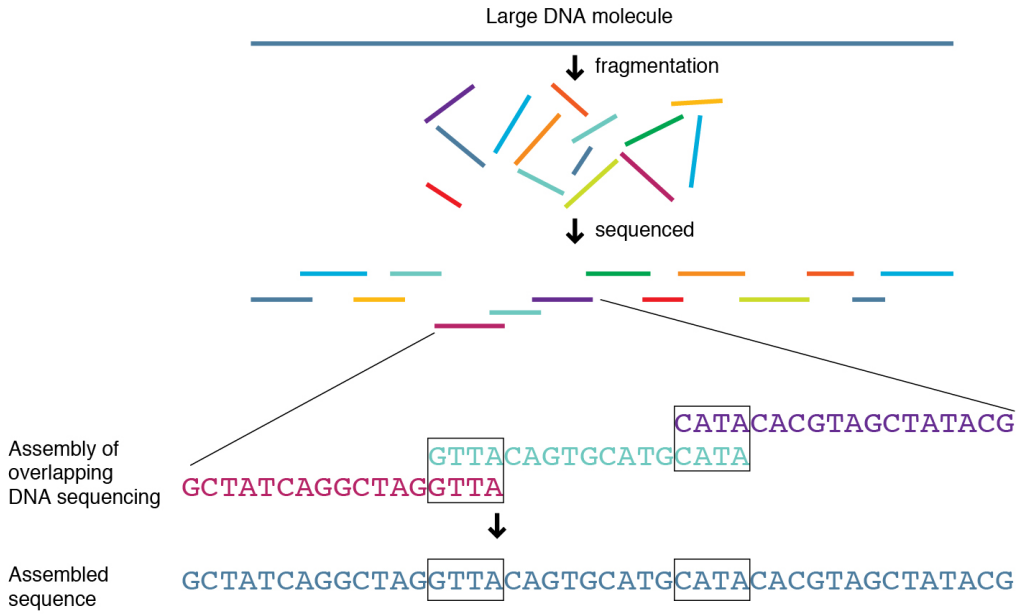

FIGURE 1.6 – Séquençage de Shotgun [5]

La méthode de séquençage Shotgun comporte les étapes suivantes :

1. Extraction de l'ADN à partir de l'ensemble ou d'une partie de l'organisme étudié, à l'aide d'un protocole choisi parmi les nombreux protocoles existants.
2. Découpage du génome en fragments qui sont insérés dans un vecteur tel qu'un BAC ou un YAC.
3. Clonage de l'ADN et séquençage des extrémités du clone.
4. Subdivision du cosmide ou du BAC cloné en fragments plus petits.
5. Sélection aléatoire de clones pour séquençage.
6. Collecte des données brutes du séquenceur.
7. Reconstruction des séquences contigües par recouvrement.
8. Assemblage de la séquence en utilisant un graphe de De Bruijn.
9. Finalisation de la séquence en comblant les trous.
10. Recherche de signification phénotypique et clinique à l'aide de polymorphismes nucléotidiques, variations d'insertion/suppression, variations du nombre d'exemplaires et mutations.

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.

1.2.4 L'assemblage de fragments d'AND

L'assemblage de fragments d'ADN consiste à reconstituer ou à fusionner des fragments d'ADN provenant d'une séquence plus longue produite par un séquenceur. Cette tâche est accomplie par un logiciel informatique appelé "assembleur". Dans le domaine de l'assemblage de fragments d'ADN, plusieurs termes sont couramment utilisés[10], tels que :

- **Lecture :**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 méthodes couramment utilisées pour l'assemblage de l'ADN : l'assemblage par cartographie (mapping) et l'assemblage de novo.

1.2.4.1 Assemblage par cartographie (Mapping)

L'assemblage par cartographie se base sur la comparaison de fragments obtenus lors de la phase de séquençage avec une séquence ADN connue. Le but est de trouver la position qui présente le plus grand chevauchement possible avec le génome de référence pour chaque fragment. Cela revient à rechercher la correspondance la plus précise entre les fragments de séquençage et le génome de référence connu, afin de reconstruire le génome complet à partir de séquences brutes fragmentées[10].

1.2.4.2 Assemblage de novo

L'assemblage de novo est la technique de reconstruction du génome complet sans avoir recours à un génome de référence. Cette méthode est particulièrement utile lorsque l'ADN provient d'une source inconnue, telle que des cheveux trouvés sur une scène de crime. Elle implique la vérification des chevauchements entre les fragments pour reconstruire le génome complet. Cependant, trouver la solution exacte en comparant deux à deux les fragments implique un nombre exponentiel de comparaisons, rendant la tâche impossible à effectuer en un temps raisonnable. Parmi les méthodes d'assemblage de novo, l'approche OLC (Overlap-Layout-Consensus) est l'une des premières utilisées avec succès pour assembler une séquence génomique.

1.2.4.2.1 L'approche OLC (Overlap-Layout-Consensus)

L'approche OLC consiste à fusionner des séquences en se basant sur leur chevauchement et en utilisant un algorithme de consensus pour déterminer la séquence finale unique. Elle commence par fusion-

ner les séquences ayant les scores les plus élevés[10].

Les assembleurs basés sur la méthode OLC sont principalement utilisés pour assembler des lectures de longueur allant de 100 à 800 pb provenant de séquençages Sanger. Le processus d'assemblage avec cette approche se déroule en trois étapes successives[10].

- **Chevauchement (overlap) :**La phase de chevauchement a pour but de déterminer les parties des lectures qui se superposent. Cela nécessite la recherche de l'alignement adéquat ou le plus long entre la fin d'une lecture et le début d'une autre. La motivation de la recherche de chevauchements entre paires de lectures est basée sur l'hypothèse que deux lectures qui se chevauchent suffisamment se trouvent très probablement l'une à côté de l'autre dans la séquence cible.
- **Alignement :**La phase de disposition (layout) se concentre sur la recherche d'un ordonnancement plausible des fragments en se basant sur les chevauchements identifiés. C'est la phase la plus complexe car la décision d'assembler deux fragments dépend de leur chevauchement qui peut être incertain en raison d'erreurs de séquençage.

  Dans la majorité des outils d'assemblage, les lectures sont associées en fonction de leur score, en commençant par celles ayant les scores les plus élevés. Si l'alignement est jugé cohérent, les deux séquences sont combinées pour former un contig[10].
- **Consensus :** Cette phase consiste à générer une séquence d'ADN basée sur les résultats de la phase d'alignement

Un exemple illustratif de l'approche OLC est donné dans la figure 1.7.

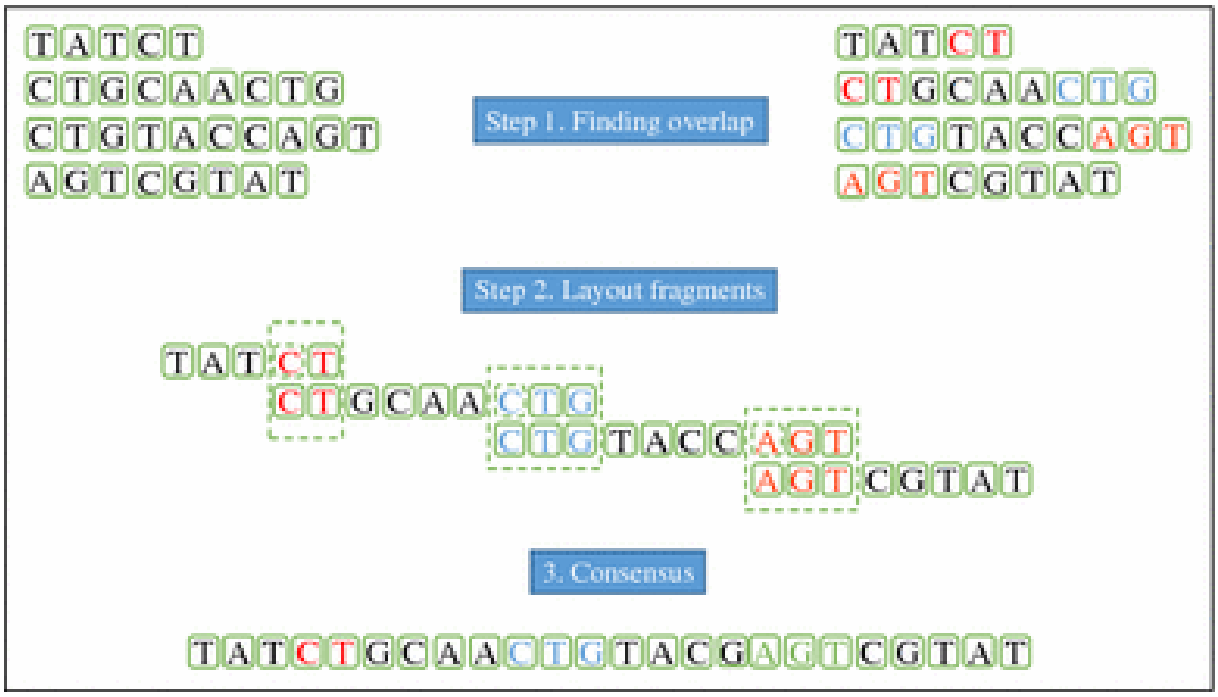

FIGURE 1.7 – La méthode OLC (Overlap-Layout-Consensus)[6]

1.2.5 Les problèmes d'assemblage

L'assemblage de fragments d'ADN peut être comparé à la tâche de reconstituer le texte d'un livre à partir de plusieurs exemplaires précédemment découpés en petits morceaux. Les défis associés à l'assemblage d'ADN comprennent :

- Toutes les machines de séquençage sont soumises à une longueur maximale de lecture imposée et peuvent produire des erreurs.
- Orientation indéterminée : lorsque la séquence d'ADN d'origine est fragmentée, l'orientation des fragments est perdue et l'extrémité à choisir devient incertaine.
- Régions répétées : pour un assembleur, deux lectures qui se superposent parfaitement proviennent de la même région génomique. Cependant, certaines séquences peuvent être présentes plusieurs fois dans un génome.
- Les assembleurs actuels ne gèrent pas de manière suffisamment efficace les séquences répétées de longue durée
- Couverture insuffisante : la répartition de la couverture est utilisée pour évaluer la qualité de la séquence consensus. Plus la couverture est élevée, moins il y a de lacunes, et le résultat est meilleur. La couverture estdéterminée par :

$$Couverture = \frac{\sum_{i=0}^{k} longueur\ du\ fragment\ i}{longueur\ du\ la\ squence\ cible} \qquad (1.1)$$

  Où K est le nombre de fragments

Nous avons vu dans cette partie que le processus d'assemblage présente plusieurs difficultés qui rendent le problème d'assemblage de fragments d'ADN comme un problème difficile. Dans la partie suivante nous allons exposer une description délaillé de l'algorithme PALS .

1.3 Partie 2 : Algorithme de recherche local PALS

Dans cette partie, nous abordons l'algorithme de recherche local PALS. Nous commençons par expliquer les méthodes heuristiques et métaheuristiques, ainsi que la définition des problèmes NP. Ensuite, nous explorons les principes des méthodes métaheuristiques les plus courantes, notamment le principe de voisinage et la recherche locale en général et l'algorithme PALS en particulier.

1.3.1 Méthodes heuristiques

Les méthodes heuristiques sont souvent utilisées pour résoudre des problèmes de grande taille pour lesquels les méthodes exactes prennent un temps exponentiel. Bien qu'elles ne garantissent pas la solution optimale, elles peuvent fournir une solution approximative assez satisfaisante. Les méthodes

heuristiques se déplacent dans l'espace de solutions en explorant les voisins de la solution courante en utilisant uniquement les informations locales (la solution courante et son voisinage). L'objectif est d'atteindre un optimum au-delà duquel aucun mouvement local n'est possible. Bien que ces méthodes soient simples et consomment peu de mémoire, elles ne dirigent généralement pas la résolution du problème vers un optimum global [10].

1.3.2 Méthodes méta heuristiques

Les méta-heuristiques sont des techniques heuristiques conçues pour résoudre des problèmes d'optimisation complexes pour lesquels les méthodes conventionnelles ne sont pas efficaces. Elles sont adaptées à chaque problème spécifique sans modification majeure de l'algorithme. La plupart des méta-heuristiques sont inspirées de la biologie, comme les algorithmes évolutionnaires, ou de l'éthologie, comme les essaims de particules et les colonies de fourmis[22].

Les méta-heuristiques sont des méthodes généralisées pour résoudre des problèmes NP, ce qui les rend indépendantes du problème considéré, contrairement aux heuristiques. Des preuves de convergence vers la solution optimale ont été établies (FAIGLE et KERN [1992]) démontrant qu'avec un temps infini, la probabilité de trouver la solution optimale augmente. Bien qu'en pratique il ne soit pas possible de laisser un temps infini, les méta-heuristiques ont démontré de très bonnes performances avec un temps fini[23].

1.3.3 Problème NP

Le problème est classé comme étant de complexité NP, ce qui signifie que même si une solution est trouvée, prouver son exactitude peut toujours être vérifié en temps polynomial (P). Si P et NP ne sont pas équivalents, alors la résolution de problèmes de complexité NP nécessite, dans le pire des cas, une recherche exhaustive[24].

Il existe deux approches pour résoudre un problème : soit en utilisant une méthode exacte pour trouver l'optimum global, soit en utilisant une méthode approximative pour trouver une solution de bonne qualité dans un délai raisonnable. Cependant, la résolution exacte d'un problème NP-difficile est souvent coûteuse car elle nécessite un temps d'exécution non polynomial en fonction de la taille de l'instance à traiter. Si l'instance du problème est de petite taille, une résolution exacte reste envisageable. Cependant, pour les instances plus grandes, une méthode approximative reste la seule option pratique pour résoudre le problème. Cette méthode consiste à explorer l'espace des solutions pour trouver une bonne solution (qui ne sera pas forcément optimale) en un temps d'exécution raisonnable[25].

1.3.4 Classification des métas heuristiques

Il existe plusieurs façons de classifier les métaheuristiques. Le schéma 1.8 vise à situer certaines des méthodes les plus populaires. Si un élément se trouve à cheval entre différentes catégories, cela indique que l'algorithme peut être classé dans l'une ou l'autre catégorie en fonction du point de vue adopté.[7].

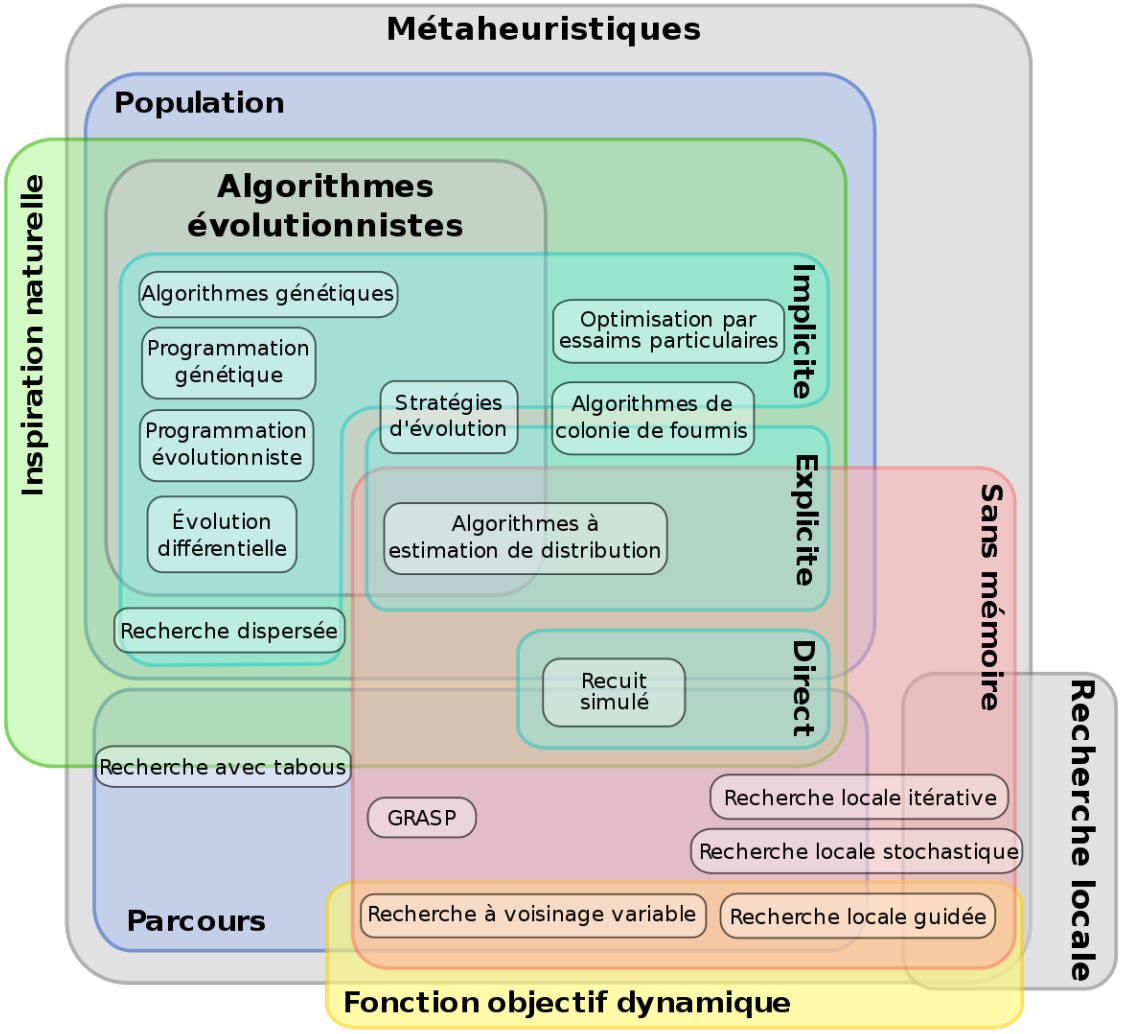

FIGURE 1.8 – Classification des Métaheuristiques [7]

1.3.5 Principe des méthodes méta heuristique les plus répondues

1.3.5.1 Voisinage

Le principe fondamental qui est largement appliqué dans la conception des méta-heuristiques est celui de la notion de voisinage. Pour chaque solution s du problème, on lui associe un sous-ensemble V(s) de solutions. Ce sous-ensemble peut être soit statique, comme dans le cas du recuit simulé, soit

dynamique et évolue avec le temps t, ou plutôt avec l'itération en cours.

Lorsqu'il s'agit d'un problème d'optimisation combinatoire non convexe et qu'un ensemble de voisinages pertinents peut être défini à l'avance, la situation se complique. Il devient alors difficile de choisir entre les différents voisinages sans faire des essais, car le voisinage n'est qu'un aspect des principes interdépendants utilisés dans l'heuristique. Le choix d'un ou plusieurs voisinages devient ainsi problématique, car il existe très peu de résultats théoriques sur la qualité d'un voisinage pour un problème spécifique.[26]

1.3.5.2 Recherche locale

La méthode de recherche locale, également connue sous le nom de descente ou d'amélioration itérative, est une famille de techniques heuristiques anciennes. Elle est largement utilisée pour résoudre des problèmes réputés pour leur grande difficulté, tels que le problème du voyageur de commerce ou la partition de graphes. Contrairement à l'approche de construction, la recherche locale opère sur des configurations complètes lors de la recherche. En somme, la recherche locale est une stratégie redoutable pour attaquer ces problèmes complexes. La figure 1.9 illustre un exemple de chemin de recherche locale. [9].

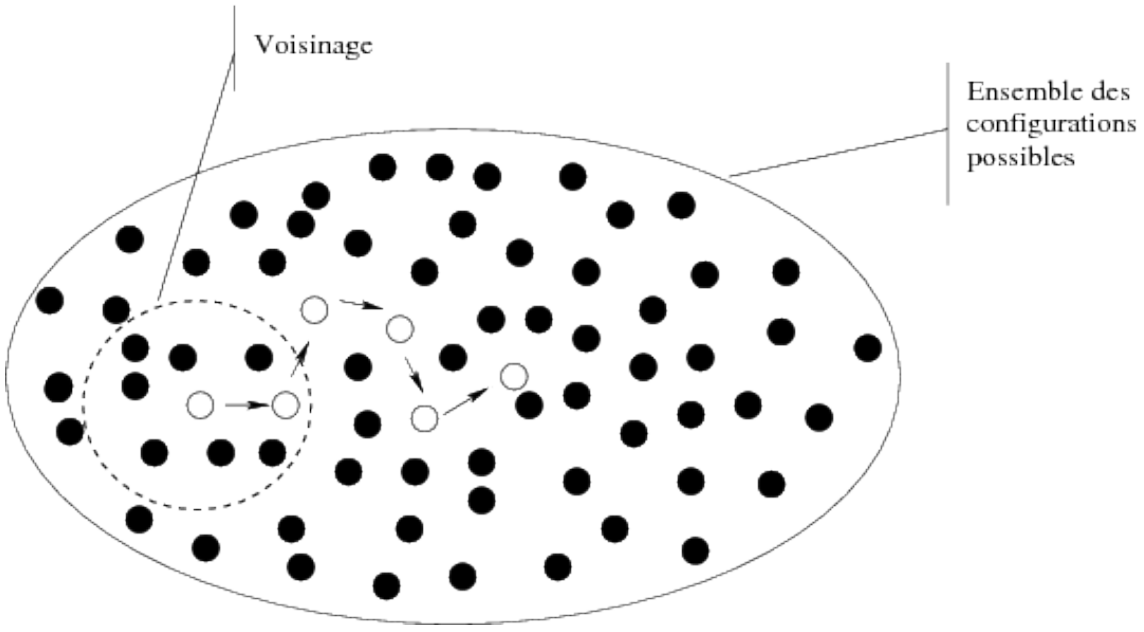

FIGURE 1.9 – Chemin de recherche locale [8]

1.3.5.3 Principe de recherche locale

Une méthode de recherche locale est un processus itératif qui repose sur deux éléments clés : un voisinage et une procédure pour exploiter ce voisinage. En particulier, la méthode commence par une configuration initiale s de l'espace de recherche X , puis elle choisit un voisin s' de s tel que la fonction objectif $f(s')$ est meilleure que $f(s)$, et remplace s par s'. Ce processus est répété jusqu'à ce qu'aucun

voisin s' ne puisse améliorer la fonction objectif, c'est-à-dire que pour tout voisin s' de s, $f(s') \geq f(s)$. À chaque itération, la méthode sélectionne donc un voisin qui améliore la configuration courante. Il existe plusieurs façons de choisir ce voisin pour améliorer la performance de la méthode.

Pour sélectionner un voisin qui améliore la configuration courante (voir la figure 1.10), il est possible d'énumérer les voisins jusqu'à ce qu'un voisin s' soit trouvé qui améliore strictement la fonction objectif (méthode de première amélioration). Comme l'espace de recherche 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 méthode de recherche locale retourne toujours un optimum local [9].

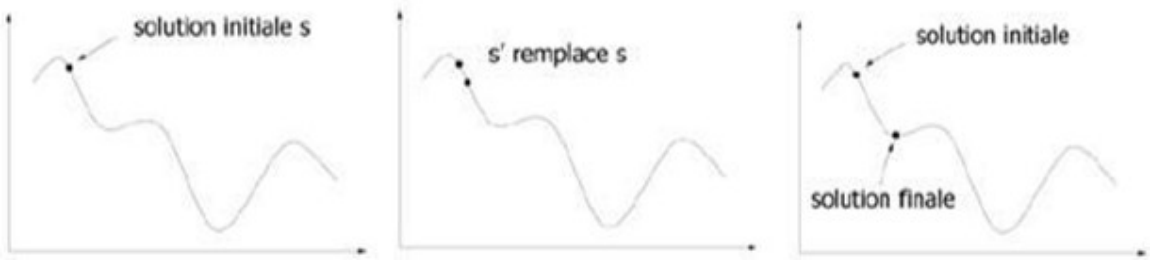

FIGURE 1.10 – Evolution d'une solution dans la méthode de recherche locale [9]

1.3.5.4 L'algorithme PALS

L'algorithme PALS, acronyme de Problem-Aware Local Search en anglais, est une méthode de recherche locale plus rapide et plus compétitive développée pour résoudre le problème d'assemblage de fragments d'ADN par Alba et Luque en 2007 [14]. Le pseudo code de cet algorithme est présenté dans la figure 1.11.

PALS est un algorithme qui améliore de manière itérative une seule solution pour l'assemblage de fragments d'ADN. La solution est représentée par une permutation d'entiers qui correspondent aux numéros des fragments, où deux fragments successifs se chevauchent. Chaque fragment est associé à un identificateur unique[10].

À chaque itération, l'algorithme remplace la solution courante par une solution de meilleure qualité dans son voisinage $N(s)$. Les solutions voisines sont générées en appliquant des mouvements à la solution courante, où un mouvement consiste à inverser l'ordre des fragments entre deux positions distinctes i et j dans la permutation (c'est-à-dire l'inversion d'une sous-permutation). L'ensemble complet des solutions voisines est défini en considérant tous les mouvements possibles pour chaque paire de positions i et j (où $1 \leq i < j \leq n$, n étant le nombre de fragments)[10].

Le point clé de PALS est l'utilisation du nombre de contigs comme critère principal pour évaluer la qualité d'un résultat d'assemblage de fragments d'ADN. Un contig est une séquence continue de fragments qui ont été assemblés avec succès[10].

En revanche, les assembleurs classiques utilisent le niveau de chevauchement entre les fragments successifs comme mesure de qualité. Ils utilisent des fonctions pour évaluer la qualité des solutions, mais cela peut conduire à des situations indésirables où une solution obtenue peut être considérée comme meilleure qu'une autre solution, alors qu'elle est en fait de mauvaise qualité, avec un grand nombre de contigs.

Dans les cas où plusieurs solutions différentes présentent le même nombre de contigs, les auteurs de PALS ont utilisé le niveau de chevauchement comme critère secondaire pour évaluer la qualité des solutions. Une solution s est considérée comme meilleure qu'une autre solution s' si et seulement si les conditions suivantes sont vérifiées[10] :

$$nContigs(s) < nContigs(s') ou (nContigs(s) = nContigs(s') et F(s) > F(s')) \tag{1.2}$$

où *nContigs* est le nombre de contigs, F (fitness) est La somme des chevauchements entre les fragments successifs.

Puisque le calcul du nombre de contigs pour chaque solution modifiée est chronophage, l'algorithme PALS utilise une évaluation incrémentale qui mesure le nombre de contigs ajoutés ou supprimés lors de la manipulation des solutions proposées. Pour chaque mouvement envisageable, l'algorithme *Calculate-Delta* calcule la variation de la longueur totale de chevauchement (Δf) ainsi que la variation du nombre de contigs (Δc) entre la solution courante et la solution modifiée (après l'application du mouvement). Ce calcul est basé sur les invariants et nécessite l'analyse des fragments impactés par le mouvement uniquement (à savoir les fragments i, j, $i-1$ et $j+1$)[10]. Le pseudo code de cet algorithme est présenté dans la figure 1.12.

Pour calculer la valeur de (Δf), on soustrait la longueur de chevauchement des fragments impactés de la solution courante de celle de la solution modifiée . En ce qui concerne (Δc), on vérifie, après l'application d'un mouvement, si un contig courant est coupé ou non ou si deux contigs ont été assemblés ou non (à travers les deux dernières conditions de l'algorithme *CalculateDelta*). Le calcul de (Δc) est basé sur un paramètre appelé "*cutoff*", qui représente la longueur minimale de chevauchement pour considérer que deux fragments adjacents sont dans le même contig[10].

À chaque itération de l'algorithme PALS, on calcule les deux critères(Δf) et (Δc) pour tous les mouvements possibles, et on stocke les mouvements acceptables dans une liste L. Seuls les mouvements qui améliorent la qualité de la solution sont pris en compte, c'est-à-dire ceux qui réduisent le nombre de contigs ou maintiennent le nombre de contigs tout en préservant le niveau de chevauchement entre les fragments adjacents . Après avoir enregistré les mouvements candidats dans la liste L, on sélectionne un mouvement spécifique et on l'applique à la solution courante pour produire une nouvelle solution

améliorée. Ce processus est répété jusqu'à ce qu'aucune amélioration ne soit possible (c'est-à-dire que la liste L soit vide)[10].

Si plusieurs mouvements ont la même valeur minimale de (Δc), on choisit celui qui a la plus grande valeur de(Δf), ce qui donne une recherche agressive. Plus précisément, un mouvement m, donné par $i, j, (\Delta c),(\Delta f)$, est considéré comme meilleur (en termes d'amélioration à apporter la solution courante) qu'un autre mouvement m', donné par $i', j', (\Delta c'), (\Delta f')$, si et seulement si les conditions suivantes sont satisfaites :

$$\Delta c < \Delta c' ou (\Delta c = \Delta c' et \Delta f > \Delta f')[10]$$

```
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  |  |  |  ⟨Δ_f, Δ_c⟩ ← calculateDelta(s, i, j) {voir algorithme }
7  |  |  |  si (Δ_c < 0) ou (Δ_c = 0 et Δ_f > 0) alors
8  |  |  |  |  L ← L ∪ ⟨i, j, Δ_f, Δ_c⟩ {Mémoriser tous les mouvements non
   |  |  |  |   détériorants}
9  |  |  |  fin
10 |  |  fin
11 |  fin
12 |  si L ≠ φ alors
13 |  |  ⟨i, j, Δ_f, Δ_c⟩ ← selectMovement(L) {Sélectionner un mouvement}
14 |  |  applyMovement(s, ⟨i, j, Δ_f, Δ_c⟩) {Améliorer la solution}
15 |  fin
16 jusqu'à pas de changement
17 retourner s
```

FIGURE 1.11 – Algorithme PALS [10].

```
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)
```

FIGURE 1.12 – Algorithme CalculateDelta [10]

1.4 Travaux connexes

Dans le domaine d'assemblage de fragments d'ADN plusieurs travaux ont été réalisés L'algorithme heuristique le plus populaire dans la littérature est appelé Problem Aware Local Search (PALS) (Alba et Luque 2007), qui est très efficace pour trouver des solutions précises et beaucoup plus rapide que les techniques existantes .Parmi les travaux qui utilisent PALS nous allons mentionner deux travaux :

1. Alba et Luque ont proposé dans (Alba et Luque, 2008) d'appliquer l'algorithme PALS comme operateur de mutation dans un GA. Dans ce schéma hybride, la population de GA est exploitée pour fournir à l'algorithme PALS de multiples configurations de départ.[14]

2. Abdelkamel Ben Ali et al, 2017. Dans cet article, ils proposent deux modifications à PALS afin d'améliorer ses performances. La première modification permet d'améliorer les solutions provisoires d'une manière plus appropriée et bénéfique. La deuxième modification permet une réduction significative des

exigences de calcul de l'algorithme sans perte de précision significative. Des expériences informatiques confirment que leurs propositions conduisent à un assembleur plus efficace et plus robuste, améliorant à la fois la précision et l'efficacité. [14]

1.5 Conclusion

Dans ce chapitre, nous avons exploré le problème d'assemblage de fragments d'ADN. En suite, nous avons introduit l'algorithme de recherche locale PALS comme une approche prometteuse pour résoudre efficacement le problème d'assemblage de fragments d'ADN. Nous avons souligné les principes fondamentaux de la recherche locale et expliqué comment l'algorithme PALS recherche itérativement une solution optimale ou quasi optimale en utilisant des techniques de recherche locale. Nous avons insisté beaucoup plus sur les détails de l'algorithme PALS que nous allons l'adopter pour résoudre le problème de fragmentation d'ADN qui sera montré dans le chapitre suivant.

Chapitre 2

Parallélisme

2.1 Introduction

Le parallélisme est un aspect essentiel de l'informatique moderne, permettant d'exploiter efficacement les ressources matérielles et d'accélérer les calculs. Dans le contexte de l'optimisation, le parallélisme joue un rôle crucial en réduisant le temps de calcul et en explorant un espace de recherche plus large. Ce chapitre examine ces aspects pour permettre une meilleure compréhension de la pertinence et des limitations du parallélisme.

2.2 Concepts clés du parallélisme

2.2.1 Définition du parallélisme

Le parallélisme désigne la capacité d'exécuter plusieurs tâches ou instructions simultanément. Il permet de diviser un problème en sous-problèmes indépendants et de les résoudre en parallèle, en utilisant plusieurs ressources de calcul , Et tout cela selon [27]. Cette approche vise à accélérer l'exécution des programmes en exploitant les capacités de traitement simultané des systèmes informatiques.

2.2.2 Avantages et Inconvénients du parallélisme

Le parallélisme offre divers avantages qu'il est important de prendre en compte lors de la conception et de l'implémentation de systèmes parallèles. Avantages du parallélisme :

2.2.2.1 Avantages du parallélisme

- **Amélioration des performances :** Le parallélisme permet d'exploiter efficacement les ressources de calcul disponibles, ce qui conduit à une accélération significative des performances des programmes.
- **Traitement de grandes quantités de données :** En parallélisant les tâches, il devient possible de traiter de vastes volumes de données de manière plus rapide et efficace.
- **Résolution de problèmes complexes :** Il offre la possibilité de résoudre des problèmes complexes en répartissant la charge de travail sur plusieurs ressources de calcul, permettant ainsi d'obtenir des résultats plus rapidement.
- **Gestion efficace des ressources :** En exploitant les ressources disponibles de manière parallèle, il permet une meilleure utilisation des processeurs et autres ressources matérielles, ce qui peut entraîner une augmentation significative de la capacité de traitement.

2.2.2.2 Inconvénients du parallélisme

- **Complexité de la programmation parallèle :** Sa mise en œuvre nécessitant une gestion minutieuse de la concurrence, de la synchronisation et de la communication entre les tâches parallèles.
- **Coûts associés à l'infrastructure :** La mise en place d'une infrastructure parallèle adéquate peut nécessiter des investissements importants en termes de matériel, de logiciels et de ressources humaines spécialisées.
- **Limites d'évolutivité :** Tous les problèmes ne bénéficient pas nécessairement d'une parallélisation efficace, et certains peuvent même rencontrer des limitations en termes d'évolutivité, ce qui signifie que l'ajout de ressources supplémentaires ne conduit pas à une amélioration proportionnelle des performances[28].

2.3 Modèles de programmation parallèle

Les modèles de programmation parallèle fournissent des abstractions et des mécanismes pour faciliter la conception et l'implémentation de systèmes parallèles. Dans cette section nous explorons les modèles les plus couramment utilisés [29] :

2.3.1 Modèle à mémoire partagée

Dans ce modèle , plusieurs processeurs partagent un espace mémoire commun, ce qui leur permet de communiquer et de coopérer en écrivant et en lisant des données de manière transparente. Ce modèle facilite la programmation parallèle en évitant les complexités liées à la communication explicite entre les processeurs. Les processeurs peuvent accéder directement aux variables partagées, ce qui permet un partage des données plus simple et une coordination efficace des tâches parallèles. Cependant, la gestion de la concurrence et de la synchronisation des accès à la mémoire partagée peut devenir un défi majeur. Différentes techniques, telles que les verrous, les sémaphores et les moniteurs, sont utilisées pour garantir l'intégrité des données partagées. Le principe de ce modèle est donné dans la figure 2.1

Le modèle à mémoire partagée, largement utilisé en programmation parallèle, peut être classé en trois catégories principales que nous trouvons dans [30]. Chacune de ces catégories présente des caractéristiques distinctes en termes d'accès à la mémoire et de gestion des données partagées. Les trois classes de la classification des systèmes de mémoire partagée sont les suivantes :

- **Accès Mémoire Uniforme (UMA) :** L'Accès Mémoire Uniforme (Uniform Memory Access en anglais) est un modèle de mémoire partagée dans lequel les processeurs ont un accès équivalent et uniforme à la mémoire partagée. Ce modèle est souvent représenté par les machines à multiprocesseurs symétriques (SMP). Lorsqu'un processeur met à jour une zone de mémoire partagée, les

autres processeurs sont automatiquement informés de la mise à jour, assurant ainsi la cohérence du cache. La gestion de la cohérence du cache est généralement réalisée au niveau matériel.

- **Accès Mémoire Non Uniforme (NUMA) :** L'Accès Mémoire Non Uniforme (Non-Uniform Memory Access en anglais) est un modèle de mémoire partagée dans lequel différents processeurs peuvent avoir des temps d'accès variables à différentes parties de la mémoire. Dans un système NUMA, chaque processeur a accès à une partie spécifique de la mémoire, généralement en se connectant à d'autres processeurs via des interconnexions. Bien que l'accès à la mémoire via ces connexions puisse être plus lent, la cohérence du cache est maintenue pour garantir l'intégrité des données partagées.
- **COMA (Cache-Only Memory Architecture) :** Le modèle COMA (Cache-Only Memory Architecture) est une variante du modèle de mémoire partagée dans laquelle la mémoire partagée est composée exclusivement de mémoire cache. Chaque processeur accède à une partie de la mémoire partagée via ses caches locaux. Lorsqu'un processeur demande l'accès à des données spécifiques, celles-ci sont migrées vers le processeur demandeur. Contrairement aux autres modèles, il n'y a pas de hiérarchie de mémoire dans le COMA, et l'espace d'adresse est constitué de tous les caches disponibles. Un répertoire de cache est utilisé pour faciliter l'accès aux caches distantes .

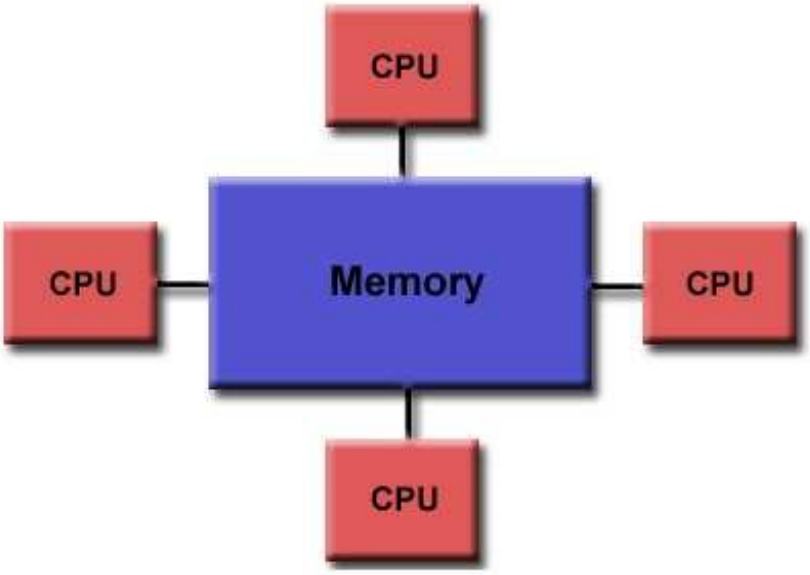

FIGURE 2.1 – Modèle à mémoire partagée [11]

2.3.2 Modèle à mémoire distribuée

Le modèle à mémoire distribuée (voir la figure 2.2) est un modèle de programmation parallèle où chaque processeur possède sa propre mémoire locale et communique avec les autres processeurs en échangeant des messages. Dans ce modèle, la mémoire n'est pas partagée entre les processeurs, ce qui nécessite une communication explicite pour échanger des données entre les différentes parties du sys-

tème. Les processeurs peuvent travailler de manière indépendante sur leurs propres tâches et accéder uniquement à leur propre mémoire locale. Cela permet une certaine flexibilité dans la conception et l'organisation des systèmes parallèles, mais nécessite une coordination et une synchronisation soigneuses pour garantir la cohérence des données partagées. Différents protocoles et algorithmes de communication sont utilisés pour faciliter les échanges de données entre les processeurs dans le modèle à mémoire distribuée.

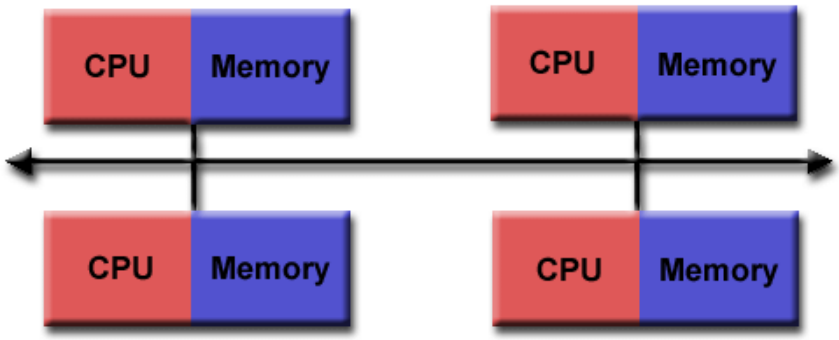

FIGURE 2.2 – Modèle à mémoire distribué [11]

2.3.3 Modèle hybride

Le modèle hybride combine les caractéristiques des modèles de mémoire partagée et de mémoire distribuée. Il regroupe les processeurs en ensembles partageant une mémoire commune pour une communication rapide à petite échelle, tout en utilisant le passage de messages pour faciliter la coordination et la communication à grande échelle entre les ensembles de processeurs. Ce modèle est largement utilisé dans les environnements de calcul à haute performance, offrant des avantages en termes de performances et d'efficacité pour les applications parallèles [29].

2.4 Type de Parallélisme

Les programmeurs ont accès à différents paradigmes de programmation parallèle, tels que le parallélisme de données, le parallélisme de tâches et le parallélisme de flux. Dans cette section, nous examinerons chacun de ces paradigmes.

2.4.1 Parallélisme de données

2.4.1.1 Principe

Le parallélisme de données est un concept essentiel en programmation parallèle qui consiste à traiter plusieurs données en même temps afin d'améliorer les performances [12]. Le principe fondamental repose sur la division des données en sous-ensembles indépendants, puis sur l'exécution simultanée de ces

sous-ensembles sur différentes unités de traitement. Cela permet d'exploiter efficacement les ressources matérielles disponibles, telles que les processeurs multi-cœurs, pour accélérer le traitement global.

Il peut être mis en œuvre de différentes manières. Une approche courante consiste à diviser les données en blocs ou en segments, puis à affecter chaque bloc à une unité de traitement distincte [31]. Chaque unité de traitement exécute les mêmes opérations sur son bloc de données assigné, ce qui permet d'accélérer le traitement global en effectuant plusieurs tâches en parallèle. Cette approche est particulièrement efficace lorsque les opérations sur les données sont indépendantes les unes des autres, car elles peuvent être exécutées simultanément sans besoin de synchronisation complexe.

Avec le parallélisme de données, il est possible d'exploiter pleinement les capacités de traitement parallèle des systèmes informatiques modernes, ce qui se traduit par des gains significatifs de performances. Cependant, il est important de prendre en compte la granularité des données, c'est-à-dire la taille des blocs de données, pour éviter les surcharges de communication et les coûts associés à la synchronisation entre les unités de traitement [32]. Une conception soigneuse et une bonne gestion du parallélisme de données sont essentielles pour obtenir des performances optimales et garantir la cohérence des résultats. La figure 2.3 présente le Parallélisme de donnée sur un processeur multi-cœurs

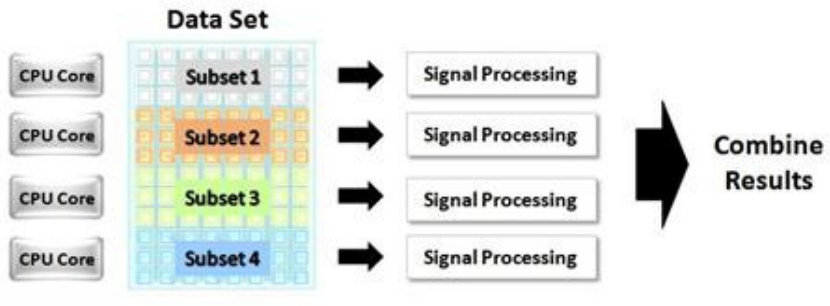

FIGURE 2.3 – Parallélisme de donnée sur un processeur multi-cœurs [12]

2.4.1.2 Avantages et inconvénients

Selon [32], nous pouvons observer que parmi les avantages et les inconvénients les plus significatifs du parallélisme de données sont les suivants :

Avantages

- **Adéquation aux structures de données :** il s'adapte naturellement aux structures de données régulières telles que les tableaux, les matrices ou les vecteurs. En divisant ces structures en parties plus petites, chaque partie peut être traitée simultanément par une unité d'exécution différente
- **Réduction du temps d'exécution :** En divisant les données en sous-ensembles et en les traitant en parallèle, le parallélisme de données permet d'accélérer le temps d'exécution global d'un programme.

- **Utilisation efficace des ressources :** En traitant simultanément plusieurs parties de données, le parallélisme de données permet une utilisation plus efficace des ressources matérielles, telles que les processeurs ou les cœurs de calcul, en les occupant pleinement.
- **Scalabilité :** Il peut être mis à l'échelle en ajoutant simplement plus de ressources de calcul, ce qui le rend adapté aux systèmes où les volumes de données à traiter sont importants et susceptibles de croître.
- **Adaptabilité aux architectures parallèles :** Il est adapté aux architectures parallèles, telles que les systèmes multiprocesseurs et les clusters de calcul.

Inconvénients

- **Dépendance des données :** Dans certains cas, des opérations sur les données dépendent des résultats des opérations précédentes [31], ce qui peut entraîner des problèmes de synchronisation et de coordination entre les tâches parallèles.
- **Overhead de communication :** Il nécessite la communication entre les différents nœuds ou threads pour échanger des données ou des résultats partiels, ce qui peut entraîner un overhead supplémentaire et une diminution des performances globales.
- **Complexité de la programmation :** car il nécessite de diviser les données, de les distribuer aux différents processeurs ou cœurs, de gérer la synchronisation et la communication, ce qui peut rendre le développement et le débogage du code plus difficiles .

2.4.2 Parallélisme de Tâches

2.4.2.1 Principe

Il permet d'exécuter simultanée de plusieurs calculs indépendants. Contrairement au parallélisme de donnée, qui parallélise le traitement de données indépendantes, le parallélisme de tâche parallélise l'exécution de tâches distinctes. Ce type de parallélisme se concentre sur les calculs effectués plutôt que sur les données manipulées. Ainsi, une même tâche peut traiter plusieurs données et plusieurs tâches peuvent partager des données, à condition que les calculs restent suffisamment indépendants[13].

Le parallélisme de tâche permet d'améliorer la réactivité d'une machine en lui permettant de traiter plusieurs tâches simultanément. Cependant, il nécessite une plus grande synchronisation que le parallélisme de donnée. Alors que le parallélisme de donnée synchronise généralement les calculs à la fin du traitement des données, le parallélisme de tâche peut nécessiter des synchronisations entre les tâches parallèles en raison de leurs interactions potentielle [33].

Contrairement au parallélisme de donnée, qui peut être réalisé au sein d'une même unité d'exécution, le parallélisme de tâche implique nécessairement l'utilisation de différentes unités d'exécution.

Les tâches parallélisées dans ce modèle sont souvent des segments de code distincts. Par conséquent, pour exploiter le parallélisme de tâche, il est nécessaire d'avoir des systèmes capables de gérer simultanément plusieurs threads d'exécution, tels que des systèmes multiprocesseurs, multi-cœurs ou SMT (Simultaneous MultiThreading). Ces systèmes présentent des similitudes avec les systèmes répartis, à l'exception du partage des ressources mémoire[13]. La figure 2.4 présente le parallélisme de tâche avec le parallélisme de données.

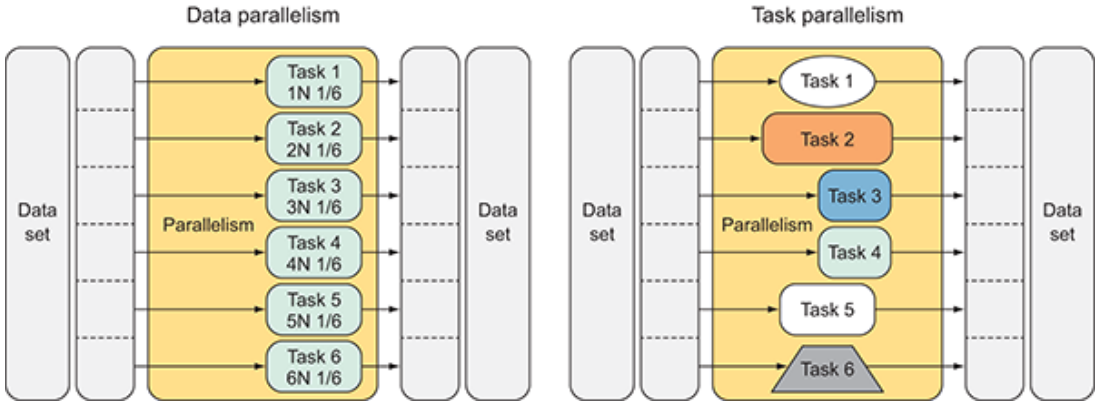

FIGURE 2.4 – Parallélisme de tâche avec parallélisme de donnée [13]

2.4.2.2 Avantages et inconvénients

Selon [32], nous pouvons observer que parmi les avantages et les inconvénients les plus significatifs du parallélisme de Tâches sont les suivants :

Avantages

- **Exécution de n'importe quel code :** La parallélisation des tâches permet d'exécuter n'importe quel code de manière parallèle, offrant ainsi une grande liberté dans la conception des programmes parallèles.
- **Adaptabilité aux différentes formes de parallélisme :** Le parallélisme de tâche peut être utilisé pour paralléliser différents types de codes et de calculs, offrant une alternative flexible aux autres formes de parallélisme.
- **Gestion des dépendances :** Le parallélisme de tâches facilite la gestion des dépendances entre les tâches, permettant de prendre en compte les relations de dépendance et d'ordonner les calculs de manière efficace.

Inconvénients

- **Complexité inhérente :** introduit une complexité supplémentaire dans le développement et la gestion du code, ce qui peut rendre la programmation plus complexe.

- **Modèle moins structuré :** Le parallélisme des tâches propose un modèle moins structuré que d'autres formes de parallélisme. Il peut être plus difficile d'identifier les tâches pouvant être parallélisées au sein d'un programme et de comprendre toutes les synchronisations qui se produisent entre elles.
- **Découpage statique et manuel des tâches :** La parallélisation des tâches nécessite un découpage manuel du code en blocs indépendants, ce qui limite la flexibilité et rend difficile l'optimisation automatique pour différentes configurations.

2.4.3 Parallélisme de flux

2.4.3.1 Principe

Le parallélisme de flux est un paradigme de programmation parallèle qui permet l'exécution simultanée de plusieurs étapes d'un traitement séquentiel, chacune traitant des données distinctes. Cela permet d'achever le traitement complet des données en les faisant progresser d'une étape à l'autre, tout en préservant l'ordre séquentiel. Les données suivent ainsi un flux continu à travers les différentes étapes, formant ainsi un flux de données caractéristique de cette forme de parallélisme [32] .

2.4.3.2 Avantages et inconvénients

Selon [32], nous pouvons observer que le parallélisme de flux présente des avantages et des inconvénients significatifs dans différentes formes. nous pouvons les classer comme suit :

Avantages

- **Parallélisation des programmes avec dépendances de données :** permet de paralléliser facilement des programmes qui ont des dépendances de données importantes. Il préserve la séquentialité du traitement parallélisé.
- **Utilisation efficace de la mémoire :** optimise l'utilisation de la mémoire, à la fois pour l'application parallélisée elle-même et pour les autres programmes du système.
- **Découpage dynamique et automatique :** Le parallélisme de flux permet un découpage dynamique et automatique du traitement séquentiel sur les données du flux.

Inconvénients

- **Performances limitées par l'étape la plus lente :** offre des performances optimales uniquement si toutes les étapes ont une durée d'exécution identique.
- **Dépendance des performances de la communication :** L'efficacité du passage à l'échelle du parallélisme de flux dépend des performances de la communication entre les unités de calcul.

2.5 Conclusion

En conclusion, le parallélisme offre de nombreuses opportunités pour améliorer les performances et l'efficacité des systèmes informatiques. En exploitant la capacité de traiter plusieurs tâches simultanément, le parallélisme permet d'accélérer les calculs, de réduire les temps d'exécution et d'optimiser l'utilisation des ressources disponibles. Cependant, il est important de prendre en compte les défis liés à la coordination des tâches parallèles, à la gestion de la concurrence et à la synchronisation des données. La compréhension de ces concepts et modèles est nécessaire pour la conception et l'implémentation de notre algorithme (PPALS) qui est la version parallèle de (PALS).

Chapitre 3

Conception et implementation

3.1 Introduction

Dans ce travail notre objectif est de proposé une version parallèle de l'algorithme de recherche locale (PALS) qui nous avons nommé (PPALS).

Dans un premier temps, nous expliquerons l'environnement de développement de notre système, les outils et les bibliothèques utilisés. Ensuite, nous examinerons l'architecture détaillé de PPALS, en décrivant son fonctionnement ainsi que les différentes phases de son exécution.

Pour évaluer notre travail nous allons appliquer notre algorithme sur divers ensembles de données. Enfin, nous comparerons les performances de PPALS à celles de l'algorithme PALS original et analyserons les résultats obtenus.

3.2 Environnement et outils de travail

3.2.1 Environnement de développement

nous avons utilisé l'IDE PyCharm qui est une plateforme de programmation Python largement utilisée qui fournit un ensemble complet d'outils pour le développement de logiciels.

FIGURE 3.1 – PyCharm

Ce programme est équipé de plusieurs fonctionnalités avancées, notamment la détection automatique des erreurs de syntaxe, la complétion de code automatique et la capacité de déboguer le code en temps réel. De plus, PyCharm est compatible avec les systèmes d'exploitation les plus courants, tels que Windows, macOS et Linux.

3.2.2 Langage de programmation et bibliothèques

Python est un langage de programmation interprété de haut niveau. Il est souvent utilisé pour le développement de logiciels, la création de scripts et l'analyse de données en raison de sa syntaxe claire et concise, ainsi que de sa grande bibliothèque standard. Python prend en charge plusieurs paradigmes de

programmation, tels que la programmation orientée objet, la programmation impérative et la programmation fonctionnelle.

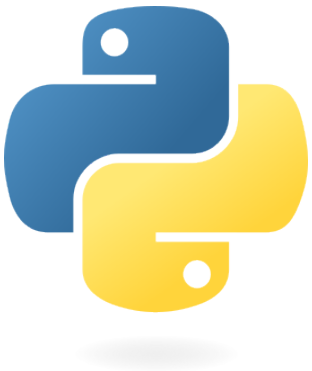

FIGURE 3.2 – Python

Les bibliothèques Python que nous avons utilisées dans ce projet comprennent :

- **Time :**Cette bibliothèque fournit des fonctions pour manipuler les dates et les heures, telles que le calcul de la durée d'exécution d'un programme.
- **Tkinter :**C'est une bibliothèque graphique pour Python, qui permet de créer des interfaces utilisateur (UI) pour les applications.
- **Filedialog :** Cette bibliothèque fournit une boîte de dialogue pour permettre à l'utilisateur de sélectionner un fichier à ouvrir ou à enregistrer.
- **Numpy :** Cette bibliothèque fournit des fonctions pour effectuer des opérations mathématiques complexes, telles que l'algèbre linéaire, la statistique et la génération de nombres aléatoires.
- **Random :** Cette bibliothèque fournit des fonctions pour générer des nombres aléatoires.
- **Multiprocessing :** Cette bibliothèque permet de créer des processus multiples pour exécuter des tâches simultanément, améliorant ainsi les performances de l'algorithme PPALS.

3.2.3 Spécifications matérielles

Les expérimentations ont été effectuées sur un ordinateur équipé des spécifications matérielles suivantes :

Système d'exploitation : Windows 10 Pro 64-bit
Processeur : Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz (4 CPUs), 2.6GHz
Mémoire RAM : 8 Go
Carte graphique : NVIDIA GeForce 840M et Intel(R) HD Graphics Family

Ces spécifications matérielles ont été suffisantes pour exécuter l'algorithme PPALS et réaliser les tests de performance sur les jeux de données sélectionnés. Il convient de noter que les performances peuvent varier en fonction de la taille des données et du degré de parallélisme utilisé.

3.2.4 Base de données utilisée

Pour notre étude, nous avons utilisé des instances d'ADN fournies par l'outil GenFrag. Cet outil a été spécialement conçu pour évaluer et comparer les performances des assembleurs en termes d'efficacité et de qualité des solutions. Il génère de manière artificielle des instances d'ADN utilisées comme référence. Les données de référence utilisées dans notre étude ont été créées par L. Michael Engle et Christian Burk [30]. Les fragments générés ont été créés avec un seuil minimal de chevauchement de 30pb, ce qui rend l'assemblage un peu plus difficile pour les instances de grande taille. Le table (Tableau 3.1) représenté les instances fournies par GenFrag.

Instance	Longueur moyenne des fragments	Nombre de fragments	Couverture	Longueur de la séquence
X60189-4 X60189-5 X60189-6 X60189-7	395 286 43 387	39 48 66 68	4 5 6 7	3835
M15421-5 M15421-6 M15421-7	398 350 383	127 173 177	5 6 7	10089
j02459-7 38525243-4 38525243-7	405 708 703	352 442 773	7 4 7	77292

TABLE 3.1 – Tableau des instances de GenFrag. [10]

GenFrag est une collection d'ensembles d'ADN composée de différentes instances de tailles variables. Chaque instance est identifiée par un nom spécifique et est stockée dans un répertoire correspondant. Par exemple, les instances X60189-4, X60189-5, X60189-6, X60189-7, M15421-5, M15421-6, M15421-7, J02459-7, 38524243-4 (BX842596-4) et 38524243-7 (BX842596-7) sont présentes dans le répertoire GenFrag.

Le répertoire associé à chaque instance contient un ensemble d'informations détaillées spécifiques à cette instance particulière. Ces informations comprennent la séquence cible d'ADN, qui fournit des détails sur le nombre de nucléotides la composant ainsi que d'autres informations pertinentes. De plus, le répertoire comprend plusieurs fichiers, notamment :

- X.dat : Cette fichier contient les fragments utilisés dans le contexte de notre problème. Les fragments sont présentés dans un format similaire à celui de fasta, ce qui facilite leur manipulation et leur utilisation dans notre algorithme PPALS.
- Un fichier Excel présentant les chevauchements entre les fragments. L'intersection de la ligne *i* et de la colonne *j* représenté le chevauchement entre les fragments *i* et *j*.
- D'autres informations concernant la matrice de chevauchement.

3.3 Architecture du système

3.3.1 Architecture globale du système

L'architecture globale du système peut être résumée en trois étapes principales, comme illustré dans la Figure 3.3 :

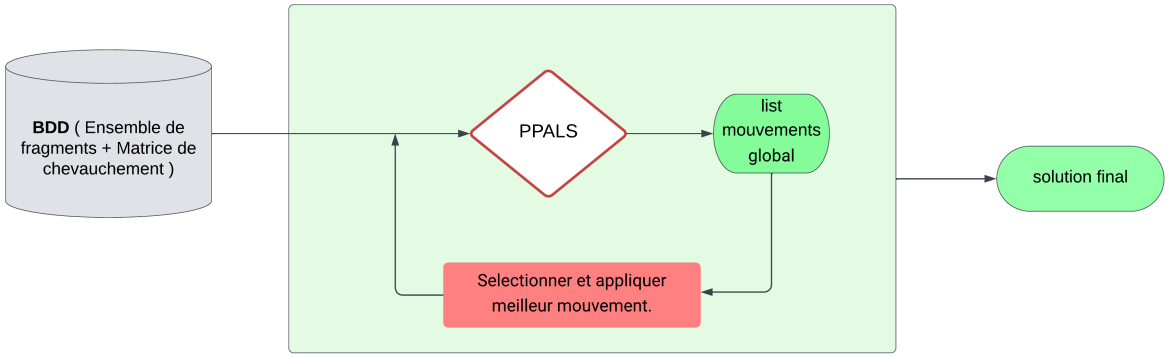

FIGURE 3.3 – L'architecture globale du système

- **BDD (Ensemble de fragments et matrice de chevauchement) :** L'algorithme PPALS prend en entrée deux types de fichiers, à savoir le fichier .dat qui représenté un fichier de fragments d'ADN, et le fichier .csv qui contient la matrice de chaînement.
- **PPALS :** L'algorithme PPALS utilise le principe de recherche locale (parallèle et itératif) pour générer une liste de mouvements globale.
- **Liste de mouvements globale :** La sortie de PPALS est une liste de mouvements potentiels qui représentent les mouvements générés après l'amélioration de la solution actuelle.
- **Sélection et application du meilleur mouvement :** Ce module sélectionne le meilleur mouvement d'après la liste de mouvements globaux (meilleur contigs et meilleure fitness), puis remplace la solution actuelle par la solution modifiée.

Le processus doit être répété jusqu'à l'obtention d'un solution final.

3.3.2 Architecture détaillée du système

L'architecture détaillée du système repose sur plusieurs étapes clés, Cette architecture est illustrée dans la Figure 3.4.

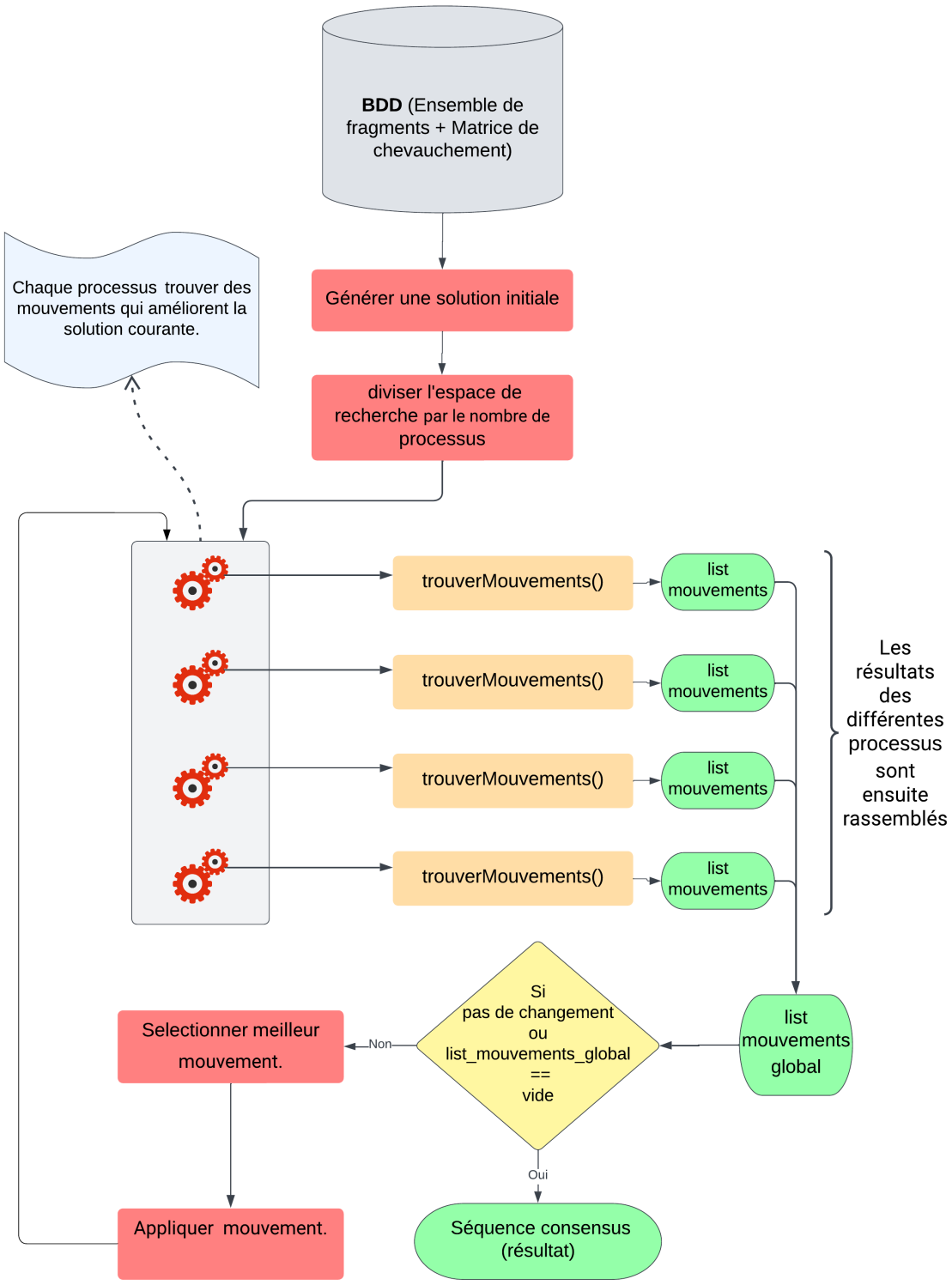

FIGURE 3.4 – L'architecture détaillée du système

- **Base de données utilisée :** L'algorithme PPALS prend en entrée un ensemble de fragments , ainsi qu'une matrice de chevauchement indiquant les chevauchements entre les fragments.
- **Générer une solution initiale :** Avant de commencer le processus parallèle, une solution initiale est générée de façon aléatoire.
- **Diviser l'espace de recherche par le nombre de processus :** l'espace de recherche (représente un morceau de fragment) est divisé par le nombre de processus disponibles(Dans notre cas, nous avons 3 processus). Chaque processus reçoit une partie spécifique de l'espace de recherche pour effectuer les mouvements nécessaires de manière indépendante.
- **Trouver les mouvements :** Chaque processus exécute la fonction **trouverMouvements()** sur sa partie de l'espace de recherche. Cette fonction permet de rechercher les mouvements potentiels qui peuvent améliorer la solution actuelle.
- **Liste de mouvements :** Chaque processus génère une liste de mouvements potentiels correspondant à sa partie de l'espace de recherche. Ces listes de mouvements sont ensuite consolidées en une liste globale de mouvements.
- **Critères d'arrêt :** Si la liste de mouvements globale est vide ou pas changement (solution actuelle = solution précédent), cela signifie qu'aucune amélioration supplémentaire n'est possible et que nous avons atteint un consensus sur la meilleure solution. Dans ce cas, l'algorithme PPALS renvoie la séquence consensus en tant que résultat final.
- **sélectionner et Appliquer le meilleur mouvement :** Si les critères d'arrêt ne sont pas satisfaits, nous sélectionnons le meilleur mouvement parmi cette liste. Ce meilleur mouvement est ensuite appliqué à la solution actuelle, ce qui met à jour et améliore la solution courante.

3.4 Implémentation

3.4.1 Importation et préparation des données

3.4.1.1 Importer le fichier de fragments

La Figure 3.5 représente un morceau du fichier de fragments.

```
>frag0000
gatgaacctgtacggcttccacggtggccag
>frag0001
ctcctgtggcctttcctatccctcaccctga
>frag0002
cacccacactcactcacctgtgacgcccacg
>frag0003
tgaacctgtacggcttccacggtggccagcg
>frag0004
ccaggagaggctcctcgggggggccctggggc
>frag0005
```

FIGURE 3.5 – Morceau du fichier de fragments

nous utilisons la fonction telecharger_fragment(fragments_fichier). Cette fonction permet de lire le contenu du fichier de fragments et de le stocker dans list ,La fonction est présenté dans La Figure 3.16 .

```
def telecharger_fragment(fragments_fichier):
    fragments = []
    with open(fragments_fichier) as f:
        for line in f:
            if not line.startswith(">"):
                fragments.append(line.strip())
    return fragments
```

FIGURE 3.6 – Fonction telecharger_fragment

3.4.1.2 Importer la matrice de chevauchement

Les fichiers de chevauchement sont généralement stockés dans des fichiers de type ".csv". La Figure 3.7 représente Matrice de chevauchement

A1 # Generated by ../matriz.pl date: Tue Sep 17 09:53:52 2013

	A	B	C	D	E	F	G	H	I	J	K	L
1	# Generated by ../matriz.pl date: Tue Sep 17 09:53:52 2013											
2	# pairs file: pairs_conservative.txt symetric matrix 68 fragments											
3	0	11	43	258	10	12	11	13	190	43	11	14
4	11	0	13	11	8	11	411	13	11	13	298	23
5	43	13	0	38	264	192	13	54	39	285	10	39
6	258	11	38	0	13	42	11	13	328	38	11	14
7	10	8	264	13	0	66	8	122	39	19	10	8
8	12	11	192	42	66	0	11	12	167	146	11	10
9	11	411	13	11	8	11	0	13	11	13	280	23
10	13	13	54	13	122	12	13	0	13	12	13	7
11	190	11	39	328	39	167	11	13	0	19	11	14
12	43	13	285	38	19	146	13	12	19	0	23	224
13	11	298	10	11	10	11	280	13	11	23	0	23
14	14	23	39	14	8	10	23	7	14	224	23	0
15	13	9	13	13	13	8	9	149	13	33	9	33
16	19	9	22	19	7	8	9	19	8	22	9	22
17	43	11	322	38	66	215	11	19	63	276	11	22
18	13	9	114	13	182	12	9	411	13	12	10	7
19	13	13	12	13	12	12	13	287	13	8	13	10
20	13	9	13	13	13	8	9	69	13	33	9	33

FIGURE 3.7 – Matrice de chevauchement

Pour importer ces fichiers, nous utilisons la fonction telecharger_chevauchement(matrix_fichier, nb_fragments), La fonction est présenté dans La Figure 3.8 .

```
def telecharger_chevauchement(matrix_fichier, nb_fragments):
    chevauchement = np.zeros((nb_fragments, nb_fragments), dtype=int)
    with open(matrix_fichier) as f:
        f.readline()
        f.readline()
        for i, line in enumerate(f):
            chevauchement[i] = list(map(int, line.strip().split(",")))
    return chevauchement
```

FIGURE 3.8 – Fonction telecharger_chevauchement.

3.4.2 Implémentation de l'algorithme PPALS

Dans cette section, nous décrirons l'implémentation de l'algorithme PPALS .

3.4.2.1 Génération de la solution initiale

la fonction "genererInitialeSolution(nb_fragments)" prend en entrée le nombre de fragments pour générer une solution initiale aléatoire . La fonction est présenté dans La Figure 3.9 .

```python
def genererInitialeSolution(nb_fragments):
    solution = list(range(1, nb_fragments + 1))
    random.shuffle(solution)
    return solution
```

FIGURE 3.9 – Function genererInitialeSolution

3.4.2.2 PPALS

- Utilisation de la bibliothèque "multiprocessing" pour paralléliser l'exécution de l'algorithme PPALS.
- La fonction "ppals(nb_fragments, chevauchement, cutoff=30, nb_procs=4)" prend en entrée le nombre de fragments, la matrice chevauchement, le seuil de coupure (cutoff) et le nombre de processus (nb_procs).
- PPALS crée un pool de processus en utilisant la classe multiprocessing.Pool. Le nombre de processus est défini par La variable "nb_procs" , comme indiqué sur la figure suivante :

```python
import multiprocessing

pool = multiprocessing.Pool(nb_procs)
```

FIGURE 3.10 – Création d'un pool de processus

3.4.2.3 Division de l'espace de recherche

- La division de l'espace de recherche est réalisée à l'aide de la fonction "diviser_frag(nb_fragments, nb_procs)". Cette fonction prend en entrée le nombre de fragments et le nombre de processus utilisé.
- La fonction retourne les indices de début et de fin pour chaque partie d'espace de recherche, qui seront utilisés par les processus individuels pour déterminer les parties qu'ils vont traiter.

```python
def diviser_frag(nb_fragments, nb_procs):
    taille_piece = nb_fragments // nb_procs
    reste = nb_fragments % nb_procs
    debut_indices = [i * taille_piece + min(i, reste) for i in range(nb_procs)]
    fin_indices = [(i + 1) * taille_piece + min(i + 1, reste) for i in range(nb_procs)]
    fin_indices[-1] -= 1
    return debut_indices, fin_indices
```

FIGURE 3.11 – function diviser_frag

3.4.2.4 Trouver des mouvements

- La fonction "trouverMouvements(solution, chevauchement, debut, fin, nb_fragments, cutoff=30)".Cet fonction prend en entrée la solution : solution actuelle , chevauchement : matrice de chevauchement , debut : index de depart , fin : index de fin . Elle est utilisée pour rechercher les différents mouvements possibles.
- Elle parcourt la solution pour trouver les mouvements qui peuvent améliorer la solution.
- Ensuite, les mouvements sont stockés dans une liste (listMouvements), qui est retournée par la fonction .La figure 3.12 présenté la fonction "trouverMouvements".

```
def trouverMouvements(solution, chevauchement, debut, fin, nb_fragments, cutoff=30):
    listMouvements = []
    for i in range(debut + 1, fin + 1):
        for j in range(i + 1, nb_fragments - 1):
            delta_f, delta_c = calculateDelta(solution, i, j, chevauchement, cutoff)
            if delta_c < 0 or (delta_c == 0 and delta_f > 0):
                listMouvements.append((i, j, delta_f, delta_c))
    return listMouvements
```

FIGURE 3.12 – function trouverMouvements

3.4.2.5 Application du mouvement

La fonction "AppliquerMouvement(solution, mouvement)" permet de remplacer la solution actuelle par la solution trouvée (par permutation des indices).

```
def AppliquerMouvement(solution, mouvement):
    i, j, delta_f, delta_c = mouvement
    solution[i], solution[j] = solution[j], solution[i]
    return solution
```

FIGURE 3.13 – function AppliquerMouvement

3.4.2.6 Calculer la fitness et le nombre de contigs

Dans le cadre d'évaluation et de résultats obtenus par l'algorithme PPALS, deux fonctions sont utilisées : "calculateContigs" et "calculateFitness".

La fonction "calculateContigs" permet de compter le nombre de contigs dans une solution donnée. Cette fonction parcourt les fragments dans la solution et compter le nombre de coupures entre les fragments. Plus le nombre de contigs est faible, plus la solution est meilleure. La fonction est présenté dans La Figure 3.14 .

```
def calculateContigs(solution, chevauchement, cutoff=30):
    contigs = 1
    for i in range(len(solution) - 1):
        if chevauchement[solution[i] - 1][solution[i + 1] - 1] <= cutoff:
            contigs += 1
    return contigs
```

FIGURE 3.14 – fonction calculateContigs

La fonction "calculateFitness" permet de calculer la valeur de fitness d'une solution donnée. Elle parcourt les fragments dans la solution et accumule les mesures de chevauchement correspondantes. Plus sa valeur est élevée, plus la solution est meilleure.La fonction est présenté dans La Figure 3.15 .

```
def calculateFitness(solution, chevauchement):
    fitness = 0
    for i in range(len(solution) - 1):
        fitness += chevauchement[solution[i] - 1][solution[i + 1] - 1]
    return fitness
```

FIGURE 3.15 – function calculateFitness

3.5 Évaluation des performances de PPALS

Dans cette section, nous évaluons les performances de l'algorithme PPALS sur différents jeux de données, en utilisant un exemple spécifique : x60189_7.

Pour le jeu de données x60189_7, nous disposons des informations suivantes :

- Le nombre des Fragments = 68 , fragments cuttoff = 30

nous obtenons les résultats suivants :

- Solution : La séquence ordonnée des fragments trouvée par l'algorithme PPALS. Par exemple, [28, 16, 24, ...].
- Fitness : La valeur de fitness associée à la solution trouvée =20196.
- Le nombre des Contigs = 1 contigs.
- Durée d'exécution : Le temps d'exécuter de l'algorithme PPALS, mesuré en secondes. =1.820 secondes.

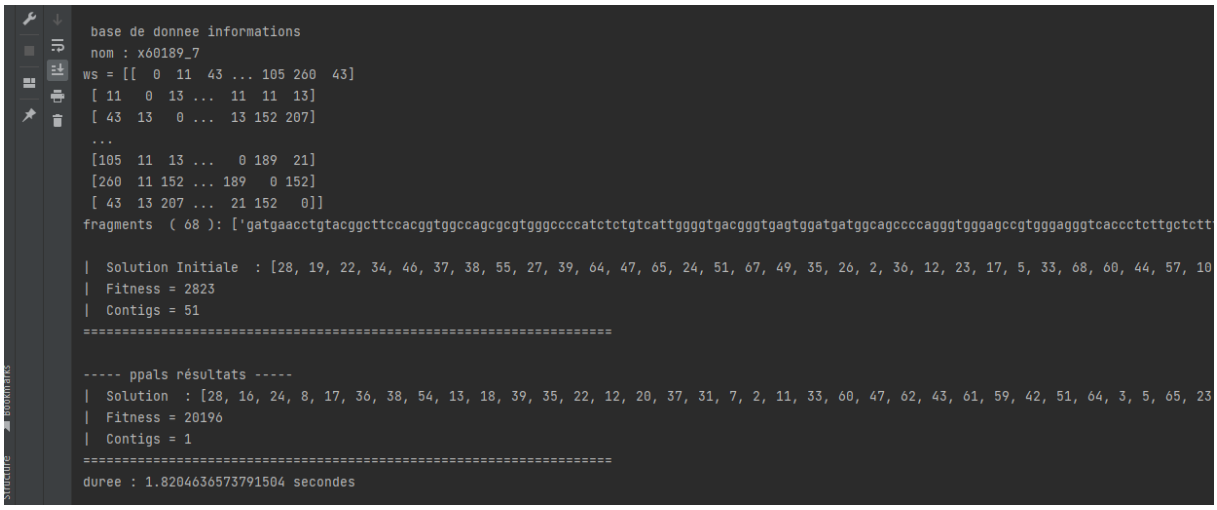

FIGURE 3.16 – résultats pour l'instance x60189_7

3.5.1 Analyse de performance de PPALS

Dans cette section, nous analysons l'impact de notre version proposée sur les performances (précision et efficacité) de PALS. Nous avons appliqué les deux algorithmes aux 4 instances de référence répertoriées dans le tableau 3.2 Nous les comparons en termes de précision (nombre de contigs et valeur de fitness) et d'efficacité (temps d'exécution).

L'instance		PALS			PPALS		
Nom d'instance	Fragments	Nombre de contigs	Fitness	Temps d'exécution(s)	Nombre de contigs	Fitness	Temps d'exécution(s)
j02459_7	352	4 :6	112732	819.04	3 :5	114423	702.51
x60189_7	68	1 :3	20612	1.99	1 :3	20204	2.77
m15421_5	127	7 :9	38156	22.41	5 :6	37905	18.00
x60189_5	48	1 :3	13237	0.430	1 :3	13321	1.03

TABLE 3.2 – Comparaison entre PALS et PPALS.

Selon le tableau , l'analyse des résultats montre que l'algorithme PPALS est plus performant en termes de temps d'exécution sur les grands jeux de données, tandis que l'algorithme PALS est plus rapide que PPALS sur les petits jeux de données. Cependant, en ce qui concerne les mesures de contigs et de fitness. Les deux algorithmes donnent des résultats similaires.

Pour une illustration visuelle des comportements des algorithmes PALS et PPALS, nous avons tracé les courbes de convergence du nombre de contigs et de la valeur de fitness.

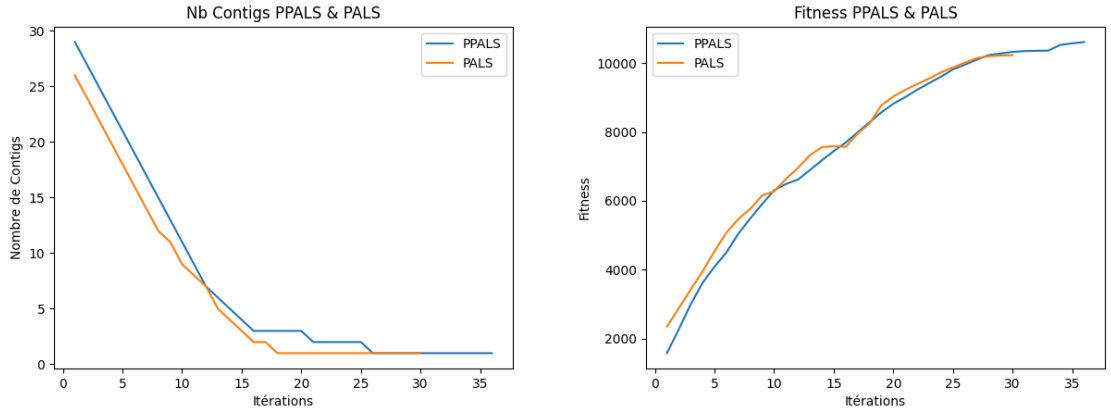

FIGURE 3.17 – Évolution du nombre de contigs et fitness lors d'une exécution pour l'instance x60189_4

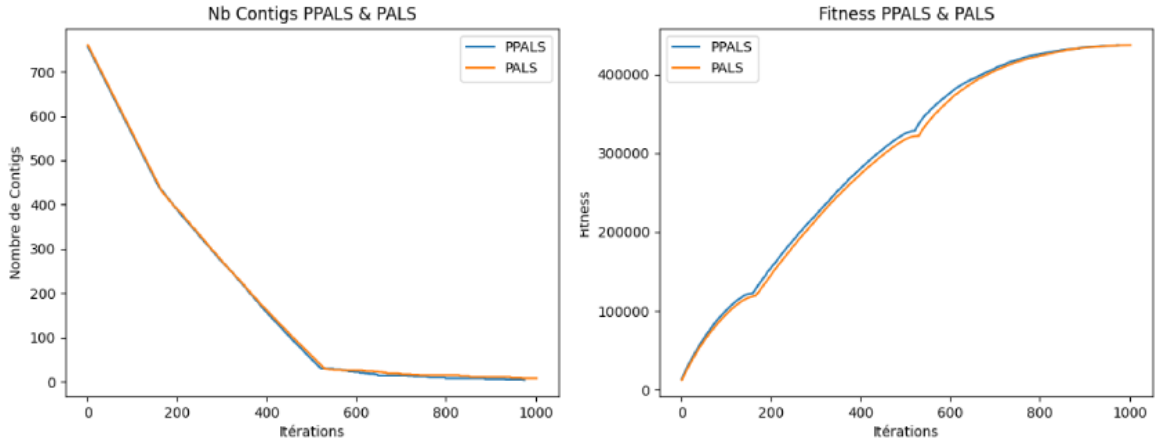

FIGURE 3.18 – Évolution du nombre de contigs et fitness lors d'une exécution pour l'instance 38524243_7

3.6 Conclusion

Ce chapitre est présenté la conception et l'implémentation de l'algorithme PPALS, une version parallèle de l'algorithme de recherche local PALS a été démontré. L'évaluation des performances a montré que PPALS est plus rapide sur les grands jeux de données tout en maintenant des résultats similaires à PALS en termes de nombre contig et la valeur fitness obtenu. PPALS constitue une alternative prometteuse pour l'assemblage de fragments d'ADN sur de grands jeux de données, offrant des gains significatifs en termes de temps d'exécution sans compromettre la qualité des résultats.

Conclusion générale

L'assemblage de fragments d'ADN est un problème d'optimisation combinatoire important en bioinformatique. Comme il s'agit d'un problème NP-difficile, des assembleurs très efficaces sont nécessaires pour traiter de grandes instances de problèmes. PALS constitue une des méthodes heuristiques les plus utilisée pour traiter ce problème, qui est très efficace pour trouver des solutions rapides et précises.

L'objectif principal de ce travail est de proposé une version parallèle de l'algorithme de recherche local PALS, nommée PPALS, afin de minimiser le temps de calcul, tout en maintenant des mesures de contig et de fitness similaires à celles de l'algorithme PALS.

L'étude a démontré que PPALS offre des performances supérieures en termes de temps d'exécution sur les grands jeux de données. Grâce à la parallélisation de l'espace de recherche et à la répartition des tâches entre différents processeurs, PPALS a pu accélérer le processus d'assemblage des fragments d'ADN. Cela est particulièrement avantageux pour les jeux de données de grande taille, où les gains en temps d'exécution sont significatifs.

Les résultats de ce projet ouvrent la voie à des nouvelles perspectives : appliquer l'algorithme sur des machines parallèles . Autres perspectives visent à combiner le nouveau PPALS avec des algorithmes métaheuristiques basés sur la population comme les algorithmes génétiques, pour résoudre avec précision des instances beaucoup plus grandes. D'autre part, l'utilisation d'instances plus grandes et de données bruitées fera également l'objet de nos futures recherches.

Bibliographie

[1] [Online]. Available : https://www.freepng.fr/png-trr85j

[2] [Online]. Available : https://www.jpboseret.eu/biologie/index.php?option=com_content&view=article&id=54&Itemid=175

[3] K. ZIANI, "Un algorithme hybride pour le problème d'assemblage de fragments d'adn," Mémoire de Master, Université BISKRA, 2022.

[4] 23/05/2023. [Online]. Available : https://www.assistancescolaire.com/eleve/1re/sciences-de-la-vie-et-de-la-terre/reviser-le-cours/1_t_02

[5] 23/05/2023. [Online]. Available : https://almerja.net/medea/images/2x_146.jpg

[6] K.-W. Huang, J.-L. Chen, C.-S. Yang, and C.-W. Tsai, "A memetic particle swarm optimization algorithm for solving the dna fragment assembly problem," *Neural Computing and Applications*, vol. 26, pp. 495–506, 2015.

[7] "Métaheuristique - classification." [Online]. Available : https://www.techno-science.net/glossaire-definition/Metaheuristique-page-3.html

[8] T. Lambert, "Hybridation de méthodes complètes et incomplètes pour la résolution de csp," Ph.D. dissertation, Université de Nantes, 2006.

[9] L. Kherbouche and Z. Oubahri, "Quelques méthodes de résolutions en optimisation combinatoire," Mémoire de Master, Université Mouloud Mammeri, TIZI OUZOU, 2017.

[10] A. BEN ALI, "Contributionsa la résolution de problemes d'optimisation combinatoires np-difficiles," Ph.D. dissertation, UNIVERSITE Mohamed Khider Biskra, 2018.

[11] A. Kiessling, "An introduction to parallel programming with openmp," in *The University of Edinburgh, A Pedagogical Seminar (accessed 24 September 2020), URL : https ://www. roe. ac. uk/ifa/postgrad/pedagogy/2009_kiessling. pdf*, 2009.

[12] "Programming strategies for multicore processing : Data parallelism." [Online]. Available : https://www.ni.com/en-lb/support/documentation/supplemental/07/programming-strategies-for-multicore-processing--data-parallelis.html

[13] "Concept : Task parallelism in category .net." [Online]. Available : https://livebook.manning.com/concept/net/task-parallelism

[14] A. Ben Ali, G. Luque, E. Alba, and K. E. Melkemi, "An improved problem aware local search algorithm for the dna fragment assembly problem," *Soft Computing*, vol. 21, pp. 1709–1720, 2017.

[15] G. M. Boratyn, A. A. Schäffer, R. Agarwala, S. F. Altschul, D. J. Lipman, and T. L. Madden, "Domain enhanced lookup time accelerated blast," *Biology direct*, vol. 7, pp. 1–14, 2012.

[16] "Bio-informatique - définition et explications." [Online]. Available : https://www.techno-science.net/glossaire-definition/Bio-informatique.html

[17] N. Chaffey, "Alberts, b., johnson, a., lewis, j., raff, m., roberts, k. and walter, p. molecular biology of the cell. 4th edn." 2003.

[18] S. Bourgoin, "Protocoles rapides pour la préparation d'adn à partir d'échantillons caractéristiques de la biologie judiciaire," Ph.D. dissertation, Université du Québec à Trois-Rivières, 2000.

[19] N. Charrat, "Diagnostic moléculaire et application en agroalimentaire : caractérisation des souches de campylobacter isolées à partir du poulet de chair au maroc." 2017.

[20] M. L. Metzker, "Sequencing technologies—the next generation," *Nature reviews genetics*, vol. 11, no. 1, pp. 31–46, 2010.

[21] C. Saad, "Caractérisation des erreurs de séquençage non aléatoires : application aux mosaïques et tumeurs hétérogènes," Ph.D. dissertation, Université de Lille, 2018.

[22] A. Drouiche, L. Cheniguel, S. Ghanem *et al.*, "Algorithmes bio-inspirés pour l'optimisation du routage dans les réseaux ad hoc," Ph.D. dissertation, univ. A/Mira. Bejaia, 2020.

[23] L. Mousin, "Extraire et exploiter la connaissance pour mieux optimiser," Ph.D. dissertation, Université de Lille, 2018.

[24] "Différence entre un problème NP-Complet et NP-Difficile." [Online]. Available : https://waytolearnx.com/2019/03/difference-entre-un-probleme-np-complet-et-np-difficile.html

[25] Y. Bouzoubaa, "Méthodes exactes et heuristiques pour l'optimisation de l'agencement d'un logement : application aux situations de handicap," Ph.D. dissertation, Université de Lorraine, 2017.

[26] L. Slimani, "Contribution à l'application de l'optimisation par des méthodes métaheuristiques à l'écoulement de puissance optimal dans un environnement de l'électricité dérégulé," Doctorat, Université de Batna 2, 2009.

[27] M. J. Quinn, *Parallel programming in C with MPI and openMP*. Dubuque, Iowa : McGraw-Hill, 2004.

[28] "Parallel computing and its advantage and disadvantage." [Online]. Available : https://www.geekboots.com/story/parallel-computing-and-its-advantage-and-disadvantage

[29] V. Louvet, "Architectures et paradigmes de programmation parallèle," in *ECOMOD 2010 – Réseau des Plasmas Froids*. Institut Camille Jordan, 2010. [Online]. Available : http://plasmasfroids.cnrs.fr/IMG/pdf/ECOMOD2010_Louvet1.pdf

[30] M. Abd-El-Barr and H. El-Rewini, *Advanced Computer Architecture and Parallel Processing.*, 2005.

[31] T. Saidani, “Optimisation multi-niveau d’une application de traitement d’images sur machines parallèles,” Ph.D. dissertation, Université Paris-Sud, 2012.

[32] T. Preud’Homme, “Communication inter-cœurs optimisée pour le parallélisme de flux,” Ph.D. dissertation, L’Université de Paris 6, 2013.

[33] “Data parallelism vs task parallelism.” [Online]. Available : https://www.tutorialspoint.com/data-parallelism-vs-task-parallelism