l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Andreea RADULESCU

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
Label européen
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et Technologies de l'Information et Mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 23 novembre 2015

Assemblage de novo de répétitions à partir de données NGS

JURY

Président :	**M. Guillaume FERTIN**, Professeur des Universités, Université de Nantes
Rapporteurs :	**M. Dominique LAVENIER**, Directeur de Recherches, CNRS Rennes
	M. Eric RIVALS, Directeur de Recherches, CNRS Montpellier
Directrice de thèse :	**M^{me} Irena RUSU**, Professeur des Universités, Université de Nantes
Co-encadrante de thèse :	**M^{me} Géraldine JEAN**, Maître de Conférences, Université de Nantes

Remerciements

Je tiens à remercier en premier lieu mes encadrants pour leur aide et pour leur soutien. Je remercie ma directrice de thèse Irena Rusu et ma co-encadrante de thèse Géraldine Jean de m'avoir donné l'opportunité d'effectuer cette thèse dans un domaine très intéressant portant à la fois sur l'informatique et la biologie. Je remercie également Guillaume Fertin d'avoir accepté de participer à l'encadrement de cette thèse. Je suis très reconnaissante envers eux pour leurs conseils, leur rigueur, leur confiance et l'encadrement expérimenté qui ont permis la réussite de cette thèse.

Je remercie également Dominique Lavenier et Eric Rivals d'avoir accepté d'être rapporteurs pour ma thèse et de participer à la soutenance. Ils ont apporté un regard extérieur sur mon travail avec des remarques pertinentes.

Je remercie Pierre Peterlongo et Mathieu Raffinot pour les échanges enrichissants à l'occasion du suivi de thèse. Leurs questions et commentaires pertinents au cours de ces trois années de thèse m'ont été très utiles pour avancer.

Je remercie également Christian Komusiewicz, avec qui j'ai eu la chance de travailler sur un sujet très enrichissant, d'avoir partagé avec moi ses connaissances et sa rigueur.

Je tiens à remercier Thomas Stützel, Manuel Lopez-Ibanez et Xavier Gandibleux pour leur encadrement lors de mon stage de recherche à l'Université Libre de Bruxelles et de m'avoir permis de découvrir le monde de la recherche.

Je suis très reconnaissante envers toute l'équipe ComBi qui m'a accueillie et soutenue dès le début de ma thèse. De même, je remercie les personnes du LINA en général pour leur accueil et leur accompagnement tout au long de la thèse. Un grand merci à chacun des doctorants (aujourd'hui docteurs pour certains) que j'ai pu côtoyer au LINA pour leur bonne humeur et leurs encouragements.

Enfin, mes plus profonds remerciements iront à mon mari, ma mère ainsi qu'au reste de ma famille. Ils ont été la source de ma motivation et de ma réussite grâce à leur soutien et encouragements indéfectibles.

Introduction

Pourquoi y a-t-il une si grande diversité de la vie ? Comment les organismes se sont-ils adaptés et ont-ils évolué au cours de leur existence ? Comment des caractéristiques spécifiques se transmettent-elles d'une génération à l'autre ? Comment comprendre et lutter contre certaines maladies ? Une molécule se trouve à la base de ces réponses : l'ADN. Depuis sa découverte, d'importantes avancées ont pu être réalisées dans de nombreux domaines comme la biologie, la médecine, l'anthropologie et la criminalistique.

L'ADN d'un organisme est regroupé dans des chromosomes, qui forment à leur tour le génome. Ce dernier encode toute l'information génétique d'un organisme par la succession dans l'ADN de quatre types de nucléotides : l'adénine, la cytosine, la guanine et la thymine. Pour connaître cette succession et l'analyser, l'ADN doit être extrait et séquencé. En séquençant un fragment d'ADN, on obtient de petites séquences de celui-ci appelées *reads* ou *lectures*. Ces petites séquences doivent, par la suite, être assemblées pour reconstituer le fragment d'ADN d'origine.

La taille d'un génome complet varie de quelques centaines de milliers de nucléotides pour les plus petits à quelques centaines de milliards de nucléotides pour les génomes les plus longs. Avec de telles longueurs, le séquençage de génomes fournit des quantités très importantes de données. De plus, depuis 2005, les méthodes de séquençage dites de nouvelle génération en ont réduit le temps et le coût de façon considérable. Ainsi, elles offrent des données à traiter se comptant en dizaines et centaines de gigaoctets. En contrepartie de l'effort réduit pour obtenir des milliers de reads, la nouvelle génération des méthodes de séquençage est responsable d'une perte d'information. Cette perte est due aux reads récupérés avec des longueurs réduites et contenant un taux d'erreurs non-négligeable. Dans ce contexte, l'assemblage de génomes est une tâche très difficile. C'est ici que l'informatique intervient pour proposer des algorithmes optimisés et adaptés à ce type de données.

Pour connaître la séquence du génome d'un organisme, le génome est séquencé plusieurs fois. Certains des reads résultants se chevauchent, la *couverture* étant le nombre moyen de reads couvrant une position arbitraire dans le génome. Le principe du processus d'assemblage est de comparer les séquences des reads et de les ordonner en fonction des chevauchements entre eux. Lorsqu'il existe une séquence d'ADN représentative pour l'espèce à laquelle l'organisme appartient, elle peut être utilisée pour aider l'assemblage des reads et obtenir plus facilement le génome de l'organisme. Cette séquence représentative est appelée *séquence de référence*. Si elle est utilisée pendant l'assemblage, on parle alors d'une approche de *mapping*, sinon l'assemblage est *de novo*. Les travaux présentés dans cette thèse s'inscrivent dans le contexte de novo.

Les algorithmes d'assemblage de novo ont fait beaucoup de progrès récemment et de nouvelles méthodes sont proposées chaque année. Ces algorithmes suivent habituellement trois grandes étapes : la corrections des reads, l'assemblage des reads chevauchants pour former des séquences plus longues appelées *contigs* et l'ordonnancement des contigs pour

construire des séquences encore plus longues appelées *scaffolds*. Pour réaliser ces étapes, les algorithmes d'assemblage de novo se basent généralement sur une des trois approches principales : l'approche *Greedy* qui a une vision locale en réalisant à chaque étape un assemblage entre les reads avec le meilleur chevauchement, l'approche *Overlap-Layout-Consensus* (OLC) qui utilise un graphe dans lequel tous les chevauchements entre les reads sont représentés pour une vision plus globale, l'approche de *de Bruijn* qui fragmente les reads en séquences chevauchantes appelées k*-mers* et construit une structure appelée *graphe de de Bruijn* pour représenter les chevauchements entre les k-mers.

Malgré ces efforts, l'assemblage de novo des génomes complets reste un défi très important, particulièrement pour les grands génomes. L'une des principales raisons est la présence de répétitions dans les génomes. Les répétitions sont des séquences ayant plusieurs copies similaires d'un motif, présentes dans un génome ou dans un fragment d'ADN quelconque. Chaque copie peut apparaître, partiellement ou intégralement, dans plusieurs reads. Comme la position d'un read dans le fragment d'ADN séquencé ne peut être déduite que par des chevauchements avec d'autres reads, les reads contenant des copies d'une répétition se chevauchent avec le même ensemble de reads et il est difficile de les localiser correctement. Cette tâche est encore plus compliquée lorsque le fragment d'ADN séquencé contient des répétitions complexes comme des répétitions incluses dans d'autres répétitions ou des répétitions plus longues que les reads. De nombreuses études montrent que, lors des assemblages de novo, les répétitions sont responsables de nombreuses parties manquantes ou mal assemblées du fragment d'ADN séquencé [104, 132]. Les assembleurs sont capables d'assembler correctement une partie des répétitions du fragment d'ADN d'origine, mais de nombreuses répétitions ne sont pas retrouvées dans les séquences résultantes à la fin de l'assemblage [117].

Le but des travaux présentés dans cette thèse est l'amélioration de l'assemblage de novo des répétitions : soit en se concentrant sur l'assemblage local d'un type spécifique de répétitions, les *répétitions en tandem*, soit en facilitant l'utilisation d'une structure appelée *graphe de de Bruijn pairé* [82], proposé pour améliorer l'assemblage global et la résolution des répétitions.

Les répétitions en tandem représentent l'un des types de répétitions les plus étudiés [55]. Elles se définissent comme des séquences composées de plusieurs copies d'un même motif situées les unes à côté des autres. Présentes dans les génomes de la plupart des organismes, les répétitions en tandem jouent un rôle important dans l'évolution des génomes, dans l'expression des gènes et dans les maladies génétiques. C'est pour cette raison que nous avons axé une grande partie des nos travaux sur l'amélioration de l'assemblage de novo des répétitions en tandem.

Notre première contribution consiste en la réalisation d'un algorithme efficace de détection des répétitions en tandem pour lesquelles les copies du motif sont identiques : les *répétitions en tandem exactes*. Dans le cas des génomes complexes contenant de nombreuses répétitions, un nombre important de répétitions en tandem exactes ne sont pas résolues par les assembleurs existants en raison de leur complexité. Nous proposons un algorithme appelé DExTaR qui a pour but de compléter l'ensemble des répétitions en tandem exactes détectées par un assembleur de novo. DExTaR est basé sur l'approche du graphe de de Bruijn en raison de la vision globale et détaillée qu'elle offre sur l'ensemble des reads à assembler. Notre algorithme analyse les parties qui n'ont pas été résolues suite à un assemblage de de Bruijn. En assemblant des contigs courts résultants, DExTaR

retrouve des répétitions en tandem exactes complexes.

Les travaux de recherche initiés avec DExTaR ont été étendus aux *répétitions en tandem approximatives*, c'est-à-dire dans le cas où les copies du motif ne sont pas toutes identiques. La nouvelle méthode, appelée MixTaR, utilise deux types de *reads* obtenus par les nouvelles méthodes de séquençage : les reads dits *courts*, utilisés également par DExTaR, et les reads dits *longs*, obtenus avec les technologies de séquençage de troisième génération apparues en 2012. Les reads longs ont, comme leur nom l'indique, des longueurs plus importantes que les reads courts, mais contiennent plus d'erreurs que ces derniers. MixTaR combine donc la bonne qualité des reads courts et la grande longueur des reads longs afin de détecter efficacement les répétitions en tandem exactes et approximatives. Contrairement à DExTaR, MixTaR n'a pas besoin d'un assemblage global préalable et effectue uniquement des assemblages locaux. L'algorithme construit son propre graphe de de Bruijn à partir des reads courts et cible les endroits où de possibles répétitions en tandem peuvent se trouver. Ensuite, MixTaR valide les motifs répétés et assemble les séquences de répétitions en tandem en utilisant à la fois les reads courts et les reads longs.

En second lieu, dans le but d'améliorer l'assemblage des répétitions, nous avons axé nos travaux sur l'assemblage basé sur le graphe de de Bruijn pairé. Ce nouveau graphe est une des structures de données proposées dans la recherche de méthodes plus efficaces pour un assemblage correct et complet avec des reads courts. Pour contourner le problème de la longueur limitée des reads courts, de nombreuses technologies de séquençage de nouvelle génération sont capables de générer des paires de reads séparés par une distance connue nommée *taille d'insert*. Ces distances sont beaucoup plus longues que les reads courts. Ce nouveau type de données appelé *reads pairés* est capable de connecter des régions contenant de longues répétitions. La plupart des assembleurs utilisent les reads pairés dans les dernières étapes de l'assemblage. L'assemblage utilisant le graphe de de Bruijn pairé propose l'inclusion de ce nouveau type de reads directement dans la structure du graphe pour une meilleure qualité d'assemblage. En contrepartie, les problèmes deviennent algorithmiquement beaucoup plus difficiles à traiter que sur les graphes de de Bruijn « classiques ». Nous avons étudié les problèmes liés à l'assemblage des génomes à partir d'un graphe de de Bruijn pairé pour proposer des algorithmes plus rapides dans le cas théorique où la taille d'insert est identique pour chaque paire de reads mais également pour le cas pratique où cette taille d'insert varie. De plus, nous décrivons un algorithme pour le problème d'assemblage dans un cas particulier de graphe de de Bruijn pairé tout en proposant une nouvelle approche pour la modélisation des graphes de de Bruijn pairés.

La suite de ce mémoire est organisée en trois parties. Dans la première partie, nous détaillons le contexte biologique et algorithmique des travaux réalisés. Nous introduisons les principales notions de génomique et les caractéristiques du processus de séquençage. Puis, nous présentons les notions d'algorithmique utilisées pour l'analyse des génomes. Nous nous focalisons sur l'assemblage des génomes en décrivant en détail les types d'algorithmes existants.

La deuxième partie est dédiée aux travaux réalisés pour la détection des répétitions en tandem. Nous exposons les types et les rôles des répétitions en tandem, ainsi que les méthodes existantes pour les détecter. Après une analyse de la qualité des assembleurs de novo et de l'adaptabilité de leurs structures de données pour la détection des répétitions en tandem, nous nous concentrons sur le graphe de de Bruijn. Nous décrivons notre méthode

DExTaR en présentant l'algorithme théorique, son adaptation aux cas pratiques ainsi que les résultats obtenus, puis les travaux effectués pour étendre la détection des répétitions en tandem avec l'algorithme MixTaR. Nous présentons aussi une analyse de la qualité des reads longs, l'adaptation de MixTaR à ce nouveau type de données et les résultats obtenus sur des données réelles et simulées.

Les travaux effectués pour faciliter l'assemblage avec le graphe de de Bruijn pairé sont détaillés dans la troisième partie du document. Nous introduisons les notions utilisées pour construire un tel graphe et pour son utilisation dans un assemblage de novo. Les travaux théoriques et les algorithmes obtenus pour rendre plus efficace l'assemblage avec le graphe de de Bruijn pairé sont également présentés.

Enfin, en conclusion de cette thèse, nous rappelons les principaux résultats obtenus et nous proposons plusieurs pistes de recherche.

Contexte scientifique

1

Notions de génomique

Toute l'information génétique d'un organisme est contenue dans son génome. En analysant le contenu des génomes, on peut comprendre l'évolution des espèces, étudier les maladies génétiques, améliorer la médecine par la mise au point de diagnostics génétiques et aider à l'avancement des enquêtes judiciaires. La science qui étudie les génomes, leur structure, leur contenu et leurs fonctions est appelée la *génomique* [42]. À l'origine, son objectif était le séquençage des génomes, mais le développement de la bio-informatique a permis l'analyse en détails des fonctions des génomes par l'assemblage de séquences génomiques et l'identification des gènes.

Ce chapitre est dédié à l'introduction des principales notions de génomique et des caractéristiques du processus de séquençage.

1.1 L'ADN

L'*acide désoxyribonucléique* ou *ADN* [124] est une molécule constituée de deux brins formés chacun d'une succession de nucléotides (voir Figure 1.1). Chaque nucléotide est composé d'une base azotée, l'*adénine* (A), la *cytosine* (C), la *guanine* (G) ou la *thymine* (T). L'ordre des nucléotides dans l'ADN encode l'information génétique permettant le développement et le fonctionnement de la majorité des organismes à l'exception de quelques virus. Chaque base est liée à un sucre, la *désoxyribose*, qui est lui-même lié à un groupement phosphate.

Les deux brins de l'ADN forment une double hélice où l'orientation de chaque brin est opposée (orientation 5'-3' ou 3'-5'). Les deux brins d'une molécule d'ADN sont *complémentaires*, une cytosine (C) fait toujours face à une guanine (G) et une adénine (A) est opposée à une thymine (T) [124]. Un *fragment d'ADN*, appelé également *séquence d'ADN*, est représenté par une succession de nucléotides, sa longueur étant exprimée en nombre de nucléotides qui le composent (*bp* pour *base pairs*).

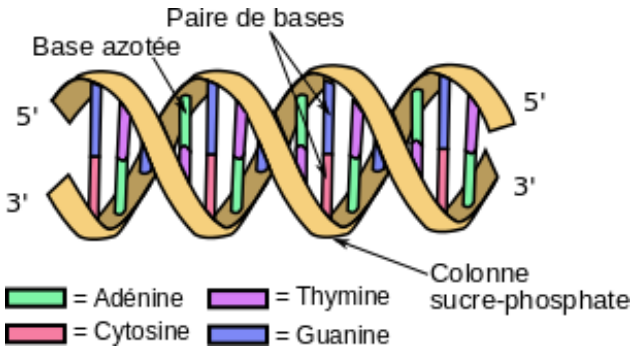


FIGURE 1.1 – Structure tridimensionnelle de la double hélice d'ADN. Un brin orienté 5'-3' est toujours apparié à un brin orienté 3'-5'. (source Wikipédia : https ://fr.wikipedia.org/wiki/Acide_désoxyribonucléique)

1.2 Constitution du génome

Génome, Chromosome, Gènes : Le *génome* regroupe l'ensemble de l'information génétique d'un organisme encodée dans son ADN [101]. Il est constitué du regroupement des différents chromosomes de l'organisme, comme le présente la Figure 1.2. Un *chromosome* est un élément microscopique, constitué de molécules d'ADN formant une chaîne double brin linéaire ou circulaire. Chaque chromosome présente des régions dites *codantes*, où se trouvent les *gènes*, et des régions appelées *non-codantes*, qui n'ont pas de fonction biologique identifiée. Les gènes sont des séquences d'ADN qui déterminent la manifestation et la transmission des caractères héréditaires.

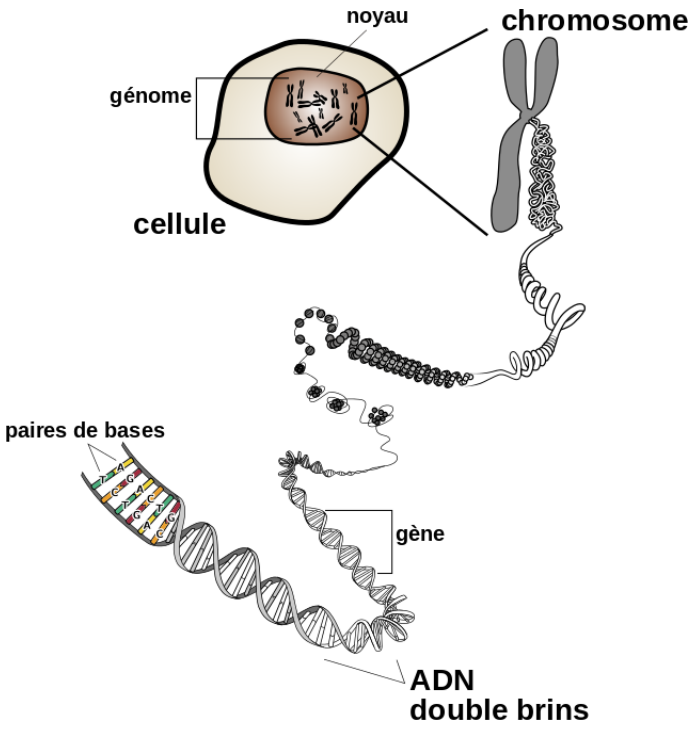


FIGURE 1.2 – Description de la structure d'un chromosome. (source Wikipédia : https ://fr.wikipedia.org/wiki/Chromosome)

Caractéristiques structurales : De nombreuses caractéristiques structurales des génomes présentent un intérêt biologique, parmi elles on retrouve la distribution des éléments répétitifs, l'étendue et la position des réarrangements des parties des génomes [42]. L'analyse de telles caractéristiques structurales est utile pour comprendre le rôle de la structure du génome dans le fonctionnement des gènes, ainsi que pour connaître les principes de base des maladies génétiques et de l'évolution des génomes.

Au cours de l'évolution, l'information génétique est transmise entre les espèces, entre les individus d'une même espèce et aussi entre les cellules d'un même individu. Pendant cette transmission, le génome peut subir des modifications appelées *mutations* [17]. Des exemples de mutations sont présentés Figure 1.3, comme la *délétion* (suppression d'un fragment d'ADN), la *duplication* (doublement du matériel génétique), l'*insertion* (addition de matériel génétique) et l'*inversion* (changement d'orientation d'une partie du chromosome). En fonction du nombre de nucléotides affectés, les mutations sont catégorisées en deux types. Les mutations *ponctuelles*, également appelées *SNP* (Single-Nucleotide Polymorphism), sont des substitutions, délétions ou insertions d'un seul nucléotide. Les mutations *chromosomiques*, quant à elles, affectent des fragments d'ADN de plusieurs nucléotides.

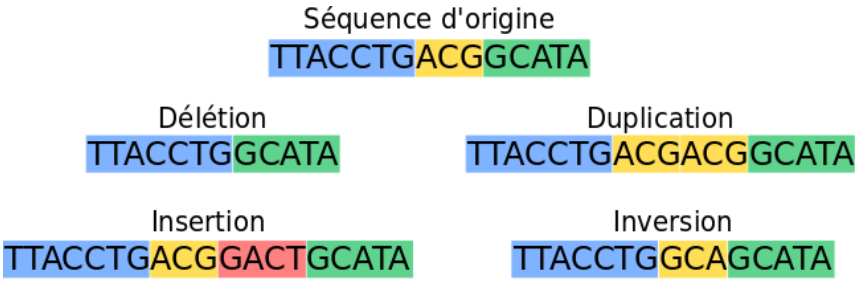


FIGURE 1.3 – Exemples de mutations génétiques : la délétion, duplication et inversion du fragment *ACG* (en jaune) et l'insertion du fragment *GACT* (en rouge).

Outre les mutations, les *séquences répétées* d'ADN sont également étudiées par les chercheurs [98, 116, 79, 133]. Les *séquences répétées*, aussi appelées *éléments répétitifs* ou encore *répétitions*, sont des motifs d'ADN qui apparaissent avec plusieurs copies dans le génome. En fonction de la distance entre les copies dans le génome, les répétitions sont soit *dispersées*, c'est-à-dire des copies réparties sur l'ensemble du génome, soit en *tandem*, quand les copies sont situées côte à côte (voir Figure 1.4). Ces séquences répétitives représentent plus de 50% du génome humain et plus de 90% du génome de plusieurs espèces comme les amibes, les criquets, ou les liliacées [42].

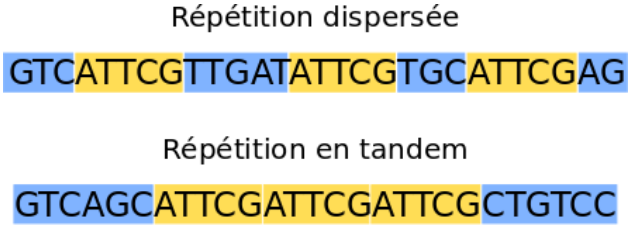


FIGURE 1.4 – Exemples de répétitions dispersées ou en tandem du motif *ATTCG* (en jaune).

1.3 Évolution du séquençage des génomes

L'un des objectifs principaux de la plupart des projets de génomique consiste à déterminer la séquence d'ADN des génomes [8]. La connaissance de la séquence d'ADN d'un génome permet à la fois l'identification des gènes ainsi que la caractérisation des nombreuses particularités structurales du génome. Cette connaissance est fondamentale pour la recherche biologique, comme, par exemple, pour l'identification et le traitement des maladies génétiques. Plusieurs étapes sont nécessaires pour déterminer la séquence d'ADN d'un génome (voir Figure 1.5). Tout d'abord, l'ADN d'un génome, appelé *fragment d'ADN cible*, doit être extrait à partir des cellules d'un individu. Le fragment d'ADN cible est découpé avec une méthode de séquençage. Le résultat du processus de séquençage est constitué de petites séquences du fragment d'ADN cible appelées *reads* ou *lectures*. Généralement ces reads appartiennent soit au brin 5'-3' du fragment d'ADN cible soit à son brin complémentaire (3'-5'), sans que l'on puisse déterminer lequel. En séquençant plusieurs fois un même fragment d'ADN cible, certains reads obtenus à la fin se chevauchent. C'est en se basant sur ces chevauchements que les reads sont ensuite *assemblés* pour reconstituer le fragment d'ADN cible. Dans cette section, nous allons présenter le principe général du séquençage qui mène à l'obtention des reads.

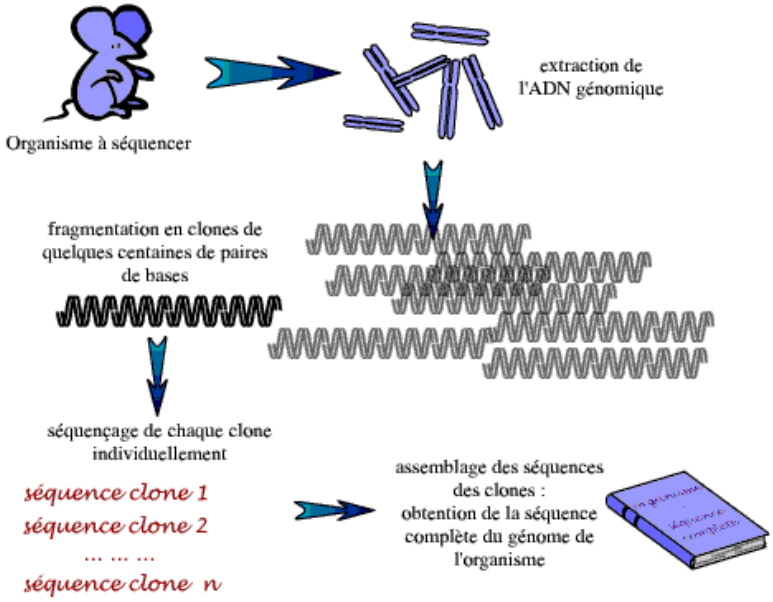


FIGURE 1.5 – Schéma générique de la construction de la séquence d'ADN d'un génome. (source http ://www.snv.jussieu.fr/vie/dossiers/genomes/)

Première génération de séquençage : Les premières méthodes de séquençage datent de 1977 : la méthode Sanger [105] est basée sur la synthèse in vitro de copies d'ADN tandis que la méthode Maxam et Gilbert [78] est basée sur la dégradation chimique des molécules d'ADN. Ces méthodes se sont avérées extrêmement efficaces dans la production de nombreux génomes complets (The International Human Genome Sequencing Consortium, 2001 ; The Mouse Genome Sequencing Consortium, 2002). Les améliorations technologiques constantes [24] et le développement de l'informatique ont permis la réalisation

de progrès scientifiques importants. Les reads résultant des procédures de séquençage, d'une longueur de 400 bp à 900 bp, sont d'une grande précision [73] (voir Tableau 1.1). Cependant, ces méthodes possèdent des inconvénients importants : leur coût est élevé et elles nécessitent l'utilisation de substances toxiques.

TABLEAU 1.1 – Caractéristiques des principales technologies de séquençage [100, 73, 19].

	Première génération	**Deuxième génération**				**Troisième génération**
Technologies	**Sanger sequencing**	**SOLiD sequencing**	**Illumina MiSeq**	**Ion Torrent sequencing**	**454**	**Pacific Biosci.**
Longueur des reads	400 à 900 bp	50+35 bp ou 50+50 bp	150 bp	400 bp	700 bp	1kbp à 20 kbp
Pourcentage d'erreurs	0,001%	0,06%	0,80%	2%	0,01%	16%
Cout (pour 1 Mb)	2400$	0,13$	0,05 à 0,15$	1$	10$	0,13 à 0,60 $

Deuxième génération de séquençage : Apparu en 2005, le séquençage de deuxième génération (*SGS* pour Second Generation Sequencing) a été développé pour réduire le temps et le coût du séquençage [2]. Ces améliorations ont été rendues possibles autant par les progrès de la biologie moléculaire que par ceux de l'informatique. Les nouvelles techniques utilisent des approches massivement parallèles et réalisent du séquençage à très haut débit. Des milliers de reads peuvent être obtenus simultanément, mais leur longueur est significativement réduite [73, 100] comme précisé dans le Tableau 1.1. De plus, le taux d'erreurs des reads est plus élevé. Les reads obtenus avec les méthodes SGS peuvent avoir jusqu'à 2% d'erreurs, tandis que la méthode de première génération, Sanger, possède un taux d'erreurs de 0,001%.

La perte d'informations due à la fois à la longueur réduite et au taux d'erreurs de reads est contrebalancée par des valeurs élevées pour la couverture. La *couverture* est un paramètre du séquençage qui représente le nombre moyen de reads couvrant une position arbitraire dans le fragment d'ADN cible. L'augmentation de la couverture permet d'augmenter la quantité d'information obtenue lors du séquençage, mais augmente également le volume de données à stocker et à analyser.

Pour répondre au problème de longueur des reads, la majorité des techniques SGS permettent la production de reads dits *pairés*. Les reads d'une paire sont séparés sur le fragment d'ADN cible par une distance approximativement connue appelée *insert*. La taille de l'insert, plus grande que la longueur d'un read, varie en général entre 500 bp et 1 kb. Ces informations supplémentaires apportées par les reads pairés facilitent l'assemblage des données SGS.

Troisième génération de séquençage : La limitation des techniques SGS concernant la longueur des reads a été abordée par les technologies Pacific Biosciences (PacBio) [16]. Dans le cadre des plateformes de séquençage de troisième génération, la technologie PacBio produit des reads d'une longueur variable allant de 1 kbp à 20 kbp. Malheureusement, l'extension de la longueur de reads s'accompagne d'une augmentation significative du taux d'erreur, qui est d'environ 16% [19]. De plus, la distribution des types d'erreurs

dans les reads obtenus avec le séquençage PacBio est différente de celle renvoyée par les précédentes technologies de séquençage. Les erreurs rencontrées dans les reads PacBio sont principalement des insertions et des délétions. Celles des reads SGS ou de première génération sont pour la plupart des substitutions. Cette différence dans le type d'erreurs entraîne un besoin d'adaptation des outils dédiés à l'assemblage de ces reads.

Par la suite, nous allons utiliser le terme *read court* pour décrire les reads obtenus avec une technologie SGS et *read long* pour les reads générés par les technologies Pacific Biosciences.

2

Assemblage des génomes

Une fois séquencé, un génome doit être reconstitué afin de pouvoir étudier sa structure et son contenu : c'est l'*assemblage*. En partant des reads obtenus par le séquençage d'un fragment d'ADN cible, les méthodes d'assemblages peuvent avoir comme objectif la reconstruction du fragment d'ADN cible entier ou seulement des parties de ce fragment présentant des caractéristiques structurales comme les répétitions ou les mutations. Dans les deux cas, les principales étapes sont l'identification des reads chevauchants, le calcul de l'ordre des reads et la déduction des séquences assemblées. Comme précisé auparavant, le fragment d'ADN cible a été extrait d'un organisme. Cet organisme appartient à une espèce pour laquelle on dispose éventuellement d'une séquence représentative appelée *séquence de référence*. Cette séquence de référence peut alors être également utilisée pour guider l'assemblage des reads appartenant au fragment d'ADN cible. Lorsque cette séquence de référence n'a pas encore été construite ou qu'elle n'est pas utilisée lors de l'assemblage, on parle, alors, d'une technique *de novo*. Les travaux de cette thèse s'inscrivent dans le contexte de novo.

Dans ce chapitre nous décrivons tout d'abord le problème général d'assemblage de novo qui vise à reconstruire tout le fragment d'ADN cible à partir de ses reads. Ensuite, nous détaillons les différents types de méthodes utilisées pour répondre au problème d'assemblage de novo. Enfin, nous présentons brièvement les techniques existantes pour l'assemblage des mutations ou des répétitions.

2.1 Assemblage des génomes entiers

L'assemblage est une tâche très complexe qui doit faire face à de nombreux types de problèmes comme les erreurs dans les reads, leur taille réduite, le manque d'information sur le brin de provenance d'un read, etc. Pour répondre à ces problèmes, une méthode d'assemblage comporte habituellement plusieurs étapes distinctes comme la correction des erreurs de reads, le regroupement des reads chevauchants en séquences plus grandes appelées *contigs*, et la construction de *scaffolds*, où les contigs sont ordonnés et séparés par des trous représentant les régions qui n'ont pas pu être retrouvées par l'assemblage.

Une autre tâche importante mais pas nécessairement identifiée comme une étape dis-

tincte est la résolution des répétitions, c'est-à-dire des séquences ayant plusieurs copies similaires dans le fragment d'ADN cible. Dans l'ensemble des reads, on perd les informations concernant la position de chaque copie d'une répétition sur le fragment d'ADN cible. En outre, chaque copie peut apparaître dans plusieurs reads, que ce soit dans son ensemble ou partiellement. Ainsi, la résolution des répétitions implique la distinction entre les reads contenant une même copie (ou des parties de celle-ci) et les reads contenant d'autres copies, similaires ou identiques, situées à des endroits différents sur le fragment d'ADN cible. Cette tâche est très difficile, surtout quand le génome contient des répétitions complexes (une répétition incluse dans une autre, ou plus longue que la longueur des reads, etc). Par conséquent, le traitement des répétitions est essentiel dans tous les assembleurs et la qualité de ce traitement a un effet majeur sur la qualité de la séquence assemblée résultante.

Il existe deux types d'approches différentes pour l'assemblage [85] : l'approche *mapping*, lorsque les reads sont assemblés à l'aide d'une séquence de référence, et l'approche *de novo*, lorsque l'assemblage est créé uniquement à partir des reads. Par la suite, nous détaillons le problème d'assemblage de novo ainsi que les solutions existantes.

La description du problème d'assemblage de novo et la synthèse des solutions existantes pour des données SGS présentées par la suite ont été réalisées en collaboration avec Géraldine Jean et Irena Rusu, et seront publiées dans [51].

2.1.1 Description du problème d'assemblage de novo

Soit $\Sigma = \{A, C, G, T\}$ l'alphabet à 4 lettres A, C, G et T représentant respectivement les nucléotides adénine, cytosine, guanine et thymine. Une séquence s constituée d'une combinaison de lettres de Σ est appelée *séquence sur l'alphabet* Σ, sa *longueur* h étant le nombre de ses caractères. Un read est donc une séquence sur l'alphabet Σ. Nous écrivons $|s| = h$ et nous disons également que s est un *h-mer*. Son i-ème caractère est noté $s[i]$ et la sous-séquence de s commençant à la position i et de longueur l est notée $s[i, l]$. Le préfixe $Pref(s, j)$ (respectivement le suffixe $Suff(s, j)$) de longueur j, avec $j > 0$, de s est la sous-séquence de s contenant les j premiers (respectivement les j derniers) caractères de s. Le *complément* $\overline{s}$ de s est obtenu en inversant l'ordre de ses caractères tout en remplaçant chaque A par un T, chaque G par un C et vice versa. En outre, étant donné $k \leq h$ nous disons qu'un k-mer x est *présent* dans s s'il existe une position i telle que $x = s[i, k]$. De façon équivalente, nous disons aussi que s *contient* x. Le *k-spectre* de s est l'ensemble $S^k(s)$ des k-mers présents dans s. Dans sa forme la plus générale, le problème d'assemblage est énoncé comme suit.

PROBLÈME DE L'ASSEMBLAGE DE NOVO

Entrée : Un ensemble non vide R de reads sur Σ, résultant du séquençage d'un fragment d'ADN cible D.

Objectif : Trouver une séquence assemblée S sur Σ contenant chaque read de R ou son complément, et de telle sorte que S soit aussi proche que possible de D.

Comme précisé auparavant, le fragment d'ADN cible possède deux brins complémentaires et le brin d'origine d'un read n'est pas connu. On ne peut donc pas en déduire son orientation. C'est pour cela que dans le PROBLÈME DE L'ASSEMBLAGE DE NOVO, on considère les reads, mais, aussi leurs compléments pour la construction de la séquence assemblée. Les reads peuvent présenter des erreurs de séquençage. Un read peut être présent dans S sous une forme corrigée, c'est-à-dire modifiée par rapport à sa séquence

résultant du séquençage.

Pour établir l'ordre des reads dans la séquence assemblée, les algorithmes d'assemblage s'appuient sur les *chevauchements* entre les reads. Pour définir ces chevauchements, on considère une mesure de similarité $sim(x, y)$ entre les reads (généralement la mesure de similarité utilisée est la Smith-Waterman [112]). Les chevauchements sont donc définis comme suit.

Définition 1 (Chevauchement)**.** *Soient r_1 et r_2 deux reads et t un nombre réel positif. On dit que r_1* chevauche *r_2 avec le poids t s'il existe un suffixe sr_1 de r_1 et un préfixe pr_2 de r_2 de telle sorte que $sim(sr_1, pr_2) = t$. Dans ce cas, on peut également dire qu'il existe un chevauchement entre r_1 et r_2. Lorsque $sr_1 = pr_2$, le chevauchement est dit* exact*. Sinon, il est dit* approximatif*. On note $w(r_1, r_2) = max\{t \mid r_1$ chevauche r_2 avec le poids $t\}$ le chevauchement de poids maximum entre r_1 et r_2 et $len(r_1, r_2) = min\{|sr_1|, |pr_2|\}$ la longueur du chevauchement.*

L'approche la plus naturelle dans la résolution du problème d'assemblage de novo est de comparer les reads de manière à identifier les chevauchements entre eux, et d'en déduire que les reads chevauchants doivent être collés dans la séquence assemblée.

Définition 2 (Extension)**.** *Soient r_1 et r_2 deux reads de longueur l avec un chevauchement exact de longueur p. L'*extension *de r_1 avec r_2 est la séquence obtenue par la concaténation de r_1 avec le suffixe de r_2 de longueur $l - p$.*

Comme les reads peuvent contenir des erreurs, les assembleurs considèrent également les chevauchements approximatifs et l'extension doit corriger ces erreurs. Dans ce cas, l'extension est faite caractère par caractère à partir de la première position du read r_1 impliqué dans le chevauchement en calculant le caractère le plus probable pour chaque position de l'extension.

Le résultat des extensions des reads est représenté par des séquences contiguës appelées contigs. En raison des nombreuses difficultés rencontrées pour effectuer un assemblage de novo, et, en particulier, la présence de répétitions dans le fragment d'ADN cible, les contigs ne couvrent pas l'ensemble du génome. Par conséquent et en pratique, les assembleurs apportent des solutions pour un problème légèrement différent du PROBLÈME D'ASSEMBLAGE DE NOVO, qui est défini comme suit :

PROBLÈME D'ASSEMBLAGE DES CONTIGS

Entrée : Un ensemble non vide R de reads sur Σ, obtenus après le séquençage d'un fragment d'ADN cible D.

Objectif : Trouver un ensemble C de séquences construites à partir de l'alphabet Σ, appelées *contigs*, contenant un nombre maximum de reads dans R ou leurs compléments, de telle sorte que chaque contig soit aussi proche que possible d'une région de D.

Les contigs sont alors les parties de la séquence assemblée, pour lesquelles l'ordre, l'orientation et la distance entre elles dans la séquence assemblée sont calculés lors de l'étape de scaffolding, qui termine la construction de la séquence assemblée.

2.1.2 Approches de l'assemblage de novo utilisant des reads SGS

Nous distinguons trois approches différentes pour l'assemblage des contigs en utilisant les données SGS. L'approche *Greedy*, dite aussi gloutonne, adopte une procédure d'extension locale des contigs en évitant le calcul des chevauchements pour toute paire de reads.

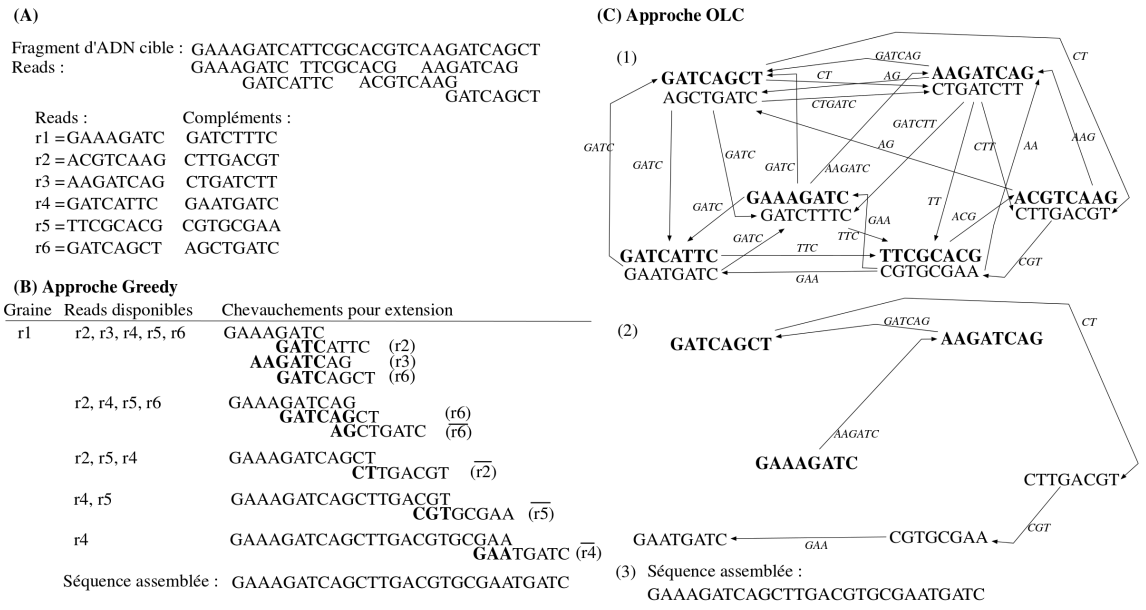


FIGURE 2.1 – Illustration des approches Greedy et OLC. (A) les reads (appartenant au même brin du fragment d'ADN cible) et leurs compléments, de longueur $l = 8$. (B) l'approche Greedy. (C) l'approche OLC. Chaque nœud contient un read et son complément. Pour simplifier l'exemple, on ne considère que les chevauchements exacts. Le chevauchement considéré pour chaque paire de reads est celui le plus long, d'une longueur d'au moins 2 caractères. (C.1) Le graphe des chevauchements est construit pendant l'étape d'Overlap. (C.2) Un chemin trouvé dans l'étape de Layout. (C.3) La séquence consensus correspondant au chemin calculé dans l'étape de consensus, qui est également, dans ce cas, la séquence assemblée.

Ses performances sont néanmoins limitées par ses choix locaux. L'approche *Overlap-Layout-Consensus* (OLC) bénéficie des informations globales obtenues par les comparaisons pour toute paire de reads, mais a besoin en contrepartie d'importantes ressources informatiques. L'approche de *de Bruijn* fragmente les reads en séquences chevauchantes appelées k*-mers*. À partir de ces k-mers et en utilisant ce que l'on appelle un graphe de de Bruijn, les chevauchements entre les reads sont facilement déduits. Cette approche est cependant très sensible aux erreurs des reads.

Approche Greedy : La méthode la plus intuitive pour la construction des contigs en utilisant les chevauchements entre les reads est l'approche Greedy. Celle-ci se base sur l'idée que plus le chevauchement entre deux reads est long, plus forte est la confirmation que les deux reads se chevauchent dans le fragment d'ADN cible. L'approche Greedy effectue successivement des choix locaux pour identifier le read qui étend le mieux un contig (lui-même étant créé par des reads chevauchants). La Figure 2.1 (A) et (B) décrit un exemple d'assemblage avec la méthode Greedy.

Comme indiqué dans la Section 2.1.1, on considère un ensemble R de reads de longueur l sur l'alphabet $\Sigma = \{A, C, G, T\}$, contenant tous les reads obtenus après séquençage, et leurs compléments. Un chevauchement entre un contig c et un read r est défini de manière similaire à un chevauchement entre deux reads (voir Définition 1). Quand un chevauchement de longueur p a été identifié, l'extension de c avec r selon ce chevauchement est obtenue par la concaténation de c avec le suffixe de r de longueur $l - p$ (voir

Définition 2).

L'Algorithme 1 présente les lignes générales d'une approche Greedy, qui suit toujours le principe d'*extension de graines* (seed-and-extend en anglais). L'algorithme calcule d'abord un ensemble S de *graines*, qui sont des reads (ou même des contigs) identifiés comme intéressants pour l'extension. Ensuite, les graines sont successivement considérées pour l'extension, chacune d'entre elles définissant un contig initial c. À noter que dans l'Algorithme 1, une fois une graine c étendue, elle et tous les reads (ainsi que leurs compléments) utilisés pour sa construction sont retirés de tous les ensembles auxquels ils appartenaient (S et R). Cette tâche est réalisée par la fonction $Supprimer(c, \overline{c}, R, S)$. L'algorithme Greedy considère d'abord les extensions possibles à l'extrémité droite de c avec des reads dans R, puis étend successivement c soit par un read ou, dans certaines versions, par un caractère à la fois. Ensuite, l'algorithme recherche de la même manière les extensions possibles à l'extrémité gauche de c en prenant le complément $\overline{c}$ de c, et non pas de R, comme il contient à la fois les reads et leurs compléments. Une fois les deux étapes d'extension finies, c est ajouté à l'ensemble C des contigs. Le scaffolding est la dernière étape. Elle utilise des reads pairés pour orienter et ordonner les contigs et ensuite déduire une séquence assemblée. Cette étape est souvent absente dans les premières versions des assembleurs.

Algorithme 1 Algorithme Greedy générique

Entrée : L'ensemble R des reads de longueur $l > 0$
Sortie : Un ensemble C de séquences assemblées

Étape 1 : Préparation des données
Calcul de l'ensemble S des graines et mise à jour de R
Étape 2 : Construction des contigs par l'extension des graines à gauche et à droite

$C \leftarrow \emptyset$;
Tant que $R \neq \emptyset$ **faire**
2.1 : Choix d'une graine c de S ; $Supprimer(c, \overline{c}, R, S)$
2.2 : Extension répétée de c, puis de $\overline{c}$, à droite
$C \leftarrow C \cup \{c\}$
fin Tant que

Étape 3 : (optionnelle) Scaffolding
Utilisation de reads pairés pour la résolution des répétitions et la connexion des contigs

Parmi les assembleurs utilisant une méthode Greedy, on peut nommer SSAKE [123], VCAKE [52], SHARCGS [26], QSRA [12], PE-Assembler [3] et GAP-Filler [89].

Les objectifs premiers des assembleurs Greedy sont la simplicité et la rapidité pour obtenir des résultats satisfaisants. Ils peuvent obtenir des résultats comparables avec ceux des assembleurs de de Bruijn ou OLC pour ce qui est de la couverture du fragment d'ADN cible par les contigs résultants [72]. Néanmoins, ils ont, en général, besoin d'une couverture plus élevée que les assembleurs de de Bruijn et produisent plus d'erreurs d'assemblage que les assembleurs de de Bruijn ou OLC [132].

Approche Overlap-Layout-Consensus : Introduite pour l'assemblage des reads Sanger [93], puis adaptée pour les données SGS, la méthode Overlap-Layout-Consensus

(OLC) propose une approche globale de l'assemblage du génome. Elle utilise un graphe de reads appelé *graphe de chevauchements* qui est construit et utilisé pendant ses trois étapes principales : *Overlap*, *Layout* et *Consensus*. Un exemple d'assemblage avec la méthode OLC est présenté sur la Figure 2.1 (C).

Étant donné l'ensemble R de reads (y compris leur compléments), le but de la première étape d'un assembleur OLC est de construire un graphe global prenant en compte les chevauchements entre les reads. Cela se fait lors de l'étape *Overlap*, où l'ensemble R est d'abord utilisé pour calculer l'ensemble O des couples (r_i, r_j) de reads tels que $r_i, r_j \in R$ et $i \neq j$, dont le chevauchement (exact ou approximatif) est significatif par rapport à certains critères spécifiques définis par une fonction de similarité $sim(.,.)$. Le choix de O a un impact important sur le résultat de l'assemblage [32]. À la fin de l'étape *Overlap*, le graphe orienté $G(R) = (R, O, w)$, appelé graphe de chevauchements de R, est construit avec l'ensemble R des nœuds et les arcs (r_i, r_j) de O. Chaque arc (r_i, r_j) du graphe est pondéré par le poids du chevauchement maximum $w(r_i, r_j)$ obtenu en utilisant une fonction de similarité $sim(.,.)$ (voir Définition 1). Bien que le calcul des chevauchements pour construire l'ensemble O ne soit pas algorithmiquement difficile, la quantité importante de reads SGS fait que le temps de calcul nécessaire pour cette étape est élevé, même lorsque les algorithmes les plus efficaces [109] sont utilisés.

Le but de la deuxième étape, appelée *Layout*, est de trouver un chemin dans le graphe de chevauchements qui utilise au moins un nœud de chaque paire de nœuds associés à un read et son complément. Cela signifie que l'orientation de chaque read dans le fragment d'ADN cible est calculée en même temps que le chemin définissant la séquence assemblée recherchée. Même lorsque l'orientation du read est connue (ce qui n'est pas le cas en pratique), ce problème est très difficile d'un point de vue algorithmique. En effet, la résolution de cette tâche revient à résoudre un problème de chemin hamiltonien, c'est-à-dire à construire un chemin contenant chaque nœud d'un graphe donné exactement une fois. Le problème du chemin hamiltonien a été prouvé NP-complet dans [39]. Par conséquent, cette étape vise plutôt l'identification des chemins linéaires (c'est-à-dire dont les nœuds internes ont un degré entrant et sortant de 1 exactement) les plus longs possible. Un read peut apparaître dans plusieurs chemins linéaires. Dans ce cas, il est considéré comme impliqué dans des répétitions.

La troisième et dernière étape est appelée *Consensus*. Elle cherche à déterminer, pour chaque chemin identifié pendant l'étape précédente, la séquence qui représente le mieux tous les reads dans le chemin.

Les assembleurs OLC sont basés sur l'approche proposée dans [58], et sont organisés comme décrit dans l'Algorithme 2. Une fois l'étape *Overlap* effectuée, les étapes *Layout* et *Consensus* peuvent être effectuées une seule fois pour construire les contigs (étape 2 suivie directement de l'étape 4 dans l'Algorithme 2). Dans ce cas, les contigs sont directement obtenus à partir des chemins trouvés dans le graphe de chevauchements $G(R)$. Sinon, les contigs peuvent être obtenus à partir de chemins trouvés dans un graphe raffiné $G'(R)$ dont les nœuds sont des séquences assemblées dites *unitigs*. Optionnellement, une étape de scaffolding termine le traitement par l'ordonnancement et l'orientation des contigs ainsi que par la résolution d'un nombre de répétitions à l'aide de reads pairés.

Parmi les algorithmes d'assemblage utilisant une méthode OLC, on retrouve Minimus [114], Edena [46], SGA [110] et CloudBrush [20].

Les assembleurs basés sur la méthode OLC permettent la réalisation d'assemblages de très bonne qualité, en particulier lorsque les reads sont longs [106, 104]. Toutefois, lorsque la taille des reads est réduite, la profondeur de la couverture doit être augmentée,

Algorithme 2 Algorithme Overlap-Layout-Consensus générique

Entrée : L'ensemble R des reads de longueur $l > 0$
Sortie : Un ensemble C de séquences assemblées

Étape 1 : Préparation des données
Préparation et correction des reads
Étape 2 : Overlap
2.1 : Construction de l'ensemble O des couples de R
2.2 : Construction du graphe de chevauchements $G(R)$
2.3 : (optionnelle) Correction d'erreurs dans $G(R)$
Étape 3 : (optionnelle) Layout-Consensus 1 : construction des unitigs
3.1 : Identification des unitigs en tant que chemins dans $G(R)$
3.2 : Construction du graphe corrigé $G'(R)$
Étape 4 : Layout-Consensus 2 : construction de l'ensemble des contigs C
Identification des contigs en tant que chemins dans $G'(R)$
Étape 5 : (optionnelle) Scaffolding
Utilisation des reads pairés pour la résolution de répétitions et la connexion des contigs

ce qui entraîne des besoins en calcul très importants. Pour les grands génomes, un échec du programme est donc possible [104, 59], et ce malgré les efforts importants qui ont été déployés pour dépasser ces problèmes. Des solutions alternatives existent, comme les assembleurs hybrides. Leur principe est de combiner deux approches, parmi lesquelles on retrouve généralement l'approche OLC. L'approche hybride Greedy-OLC repose sur la technique d'extension des graines (comme dans les approches Greedy) et recherche la meilleure extension dans un graphe local de chevauchements (comme dans les méthodes OLC) qui, parfois, n'est pas explicitement construit. C'est le cas de Taipan [107], SHORTY [47] et Telescoper [11].

Approche de de Bruijn : L'idée de représenter tous les reads comme une collection de séquences encore plus petites à l'aide d'un graphe a été proposée pour la première fois en 1995 [48]. Appliquée pour l'assemblage des reads Sanger, cette approche permet de transformer la recherche d'un chemin hamiltonien (correspondant au fragment d'ADN cible dans le modèle OLC) en une recherche d'un chemin eulérien (c'est-à-dire guidé par les arcs à la place des nœuds). Cependant, la présence de répétitions et d'erreurs dans les données rend cette approche peu efficace pour les grands génomes. Les premières solutions à ces problèmes sont présentées en 2001 [96]. Dans cette version, les reads sont corrigés avant la construction du graphe. Les propriétés auxquelles les chemins doivent satisfaire pour préserver les reads sont utilisées pour lever l'ambiguïté aux croisements des chemins. À la fin de l'assemblage, les reads pairés sont utilisés pour construire des scaffolds à partir des contigs résultants. Beaucoup d'autres assembleurs proposés depuis utilisent cette approche, que nous présentons ici dans sa forme la plus générale.

Soit R l'ensemble des reads (y compris leurs compléments), de longueur l dans l'alphabet Σ. Étant donné un entier positif $k \leq l$, tout read r de longueur l peut être coupé en $l - k + 1$ k-mers successifs respectivement à partir des positions $1, 2, \ldots, l - k + 1$ de r. Soit S^k le *k-spectre* de R. Un exemple est montré Figure 2.2 (A).

Le graphe de *de Bruijn* $G^k(R)$ est le graphe orienté défini comme suit. Son ensemble de nœuds est S^k. Un arc est construit entre les k-mers s et t (dans cet ordre) si et seulement

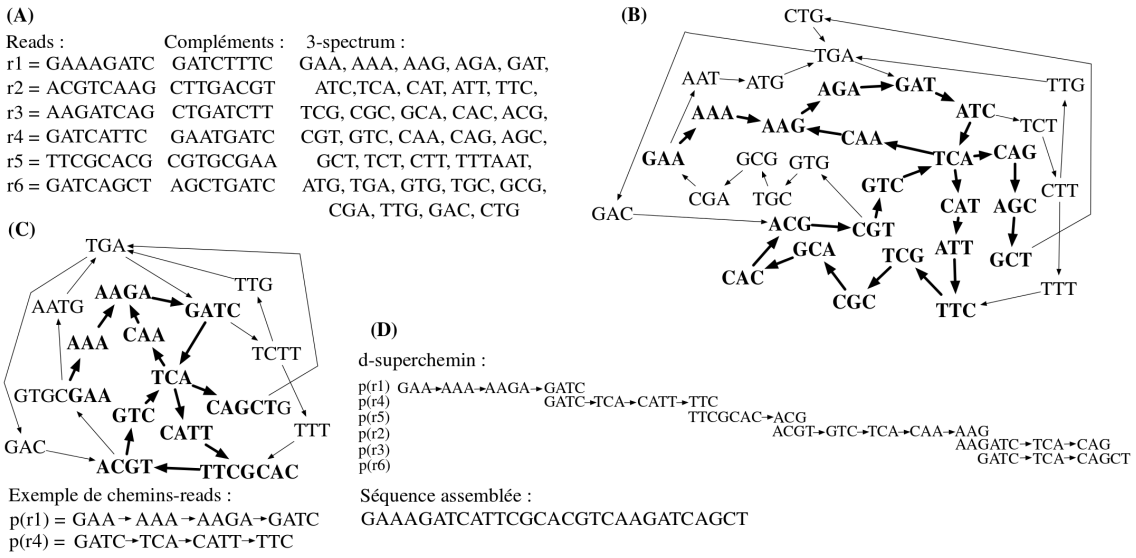


FIGURE 2.2 – Illustration de l'approche de de Bruijn, sur le même exemple que sur la Figure 2.1 pour $l = 8$ et $k = 3$. (A) L'ensemble R contenant les reads et leurs compléments et le 3-spectre $S^3(R)$ de R. (B) Le graphe de de Bruijn résultant, dans lequel le sous-graphe en gras correspond à un brin du fragment d'ADN cible et les autres k-mers appartenant au brin complémentaire. (C) Chaque chemin linéaire (sans branchement) dans (B) a été compressé dans un nœud. (D) Le d-superchemin donnant la séquence assemblée du fragment d'ADN cible.

si s et t ont un chevauchement exact de $k-1$ caractères, et s'il existe un read r qui contient le $(k+1)$-mer x produit par l'extension de s avec t, c'est-à-dire la concaténation de s avec $t[k]$.

Dans le graphe de de Bruijn, chaque chemin $(s_1, s_2), (s_2, s_3), \ldots, (s_{p-1}, s_p)$ représente une séquence d'ADN obtenue par des extensions successives de s_1 avec $s_2, s_3, \ldots s_p$. Comme précédemment, un chemin est linéaire si tous ses nœuds possèdent un degré entrant et sortant égal à 1. Le *chemin-read* correspondant à un read $r \in R$ est le chemin $(s_1, s_2), (s_2, s_3), \ldots, (s_{l-k}, s_{l-k+1})$, noté $\varsigma(r)$, tel que $s_i = r[i, k]$. Dans ce contexte, le problème d'assemblage de novo se réduit au problème défini ci-après, pour lequel nous rappelons que chaque arc dans le graphe de de Bruijn appartient à au moins un chemin-read.

PROBLÈME DU D(EMI)-SUPERCHEMIN DE DE BRUIJN

Entrée : Un entier $k > 0$, le graphe de de Bruijn $G^k(R)$ correspondant à l'ensemble R des reads, et l'ensemble $\mathcal{C}^k(R)$ des chemins-reads pour les reads dans R.

Objectif : Trouver dans $G^k(R)$ le plus court d-superchemin W de $\mathcal{C}^k(R)$, c'est-à-dire le plus court chemin dans $G^k(R)$ créé par des chemins chevauchants de $\mathcal{C}^k(R)$ et qui contient au moins $\varsigma(r)$ ou $\varsigma(\overline{r})$, pour chaque $r \in R$.

La NP-complétude du PROBLÈME DU D(EMI)-SUPERCHEMIN DE DE BRUIJN est facilement déduite de [81, 90]. L'intérêt de l'approche de de Bruijn n'est pas de réduire la complexité du problème sous-jacent mais de faciliter les calculs. Il faut toutefois noter que la plupart des algorithmes basés sur l'approche de de Bruijn n'utilisent pas les chemins-reads, les assembleurs préférant éviter les exigences de calcul qui leur sont associées. Cette information peut, jusqu'à un certain point, être remplacée par les statistiques de couverture sur chaque arc.

Les étapes générales de l'approche de de Bruijn sont données dans l'Algorithme 3. Étant donné que le graphe de de Bruijn est très sensible aux erreurs de séquençage dans les reads, ces erreurs sont corrigées par des procédures efficaces de préférence avant la construction du graphe. En outre, certains types d'erreurs peuvent être détectés après la construction du graphe. C'est le cas des chemins finissant par un nœud sans arc sortant, en raison d'erreurs de séquençage à une extrémité d'un read, et des chemins qui divergent puis convergent à cause des erreurs de séquençage au milieu d'un read.

Comme auparavant, des difficultés dans l'étape de construction des contigs sont provoquées par la présence de répétitions et en particulier par les répétitions complexes (répétitions comprises dans d'autres répétitions, répétitions en tandem, etc). Résoudre ces répétitions dans le graphe de de Bruijn, c'est-à-dire être capable de distinguer les ramifications correctes, est d'autant plus difficile lorsque des séquences plus courtes comme les k-mers sont utilisées à la place des reads. Par conséquent, la recherche d'une solution pour le PROBLÈME DU D(EMI)-SUPERCHEMIN DE DE BRUIJN passe par une simplification du graphe $G^k(R)$ en résolvant certaines répétitions. Le graphe est modifié en conséquence, de manière à obtenir dans le cas idéal un graphe divisé en deux parties, l'une correspondant au génome cible et l'autre à son complément. Afin de se rapprocher d'un tel graphe, les chemins linéaires sont très importants, car ils peuvent être contractés en un arc (ou alternativement en un nœud), sans modifier l'ensemble des solutions.

Il faut noter que cette approche solutionne le PROBLÈME DU D(EMI)-SUPERCHEMIN DE DE BRUIJN seulement dans le cas où toutes les répétitions peuvent être résolues. Très souvent dans la pratique, des répétitions non résolues persistent, ce qui implique qu'une partie du graphe ne peut pas être désambiguïsée sous la forme d'un chemin linéaire. Par conséquent, la résolution du PROBLÈME DU D(EMI)-SUPERCHEMIN DE DE BRUIJN n'est pas garantie par l'algorithme proposé, qui ne trouve que des parties du d-superchemin recherché, résolvant ainsi le PROBLÈME D'ASSEMBLAGE DES CONTIGS plutôt que le PROBLÈME D'ASSEMBLAGE DE NOVO.

Algorithme 3 Algorithme de de Bruijn générique

Entrée : L'ensemble R de reads de longueur $l > 0$ et un entier $k > 0$ qui définit la longueur de k-mers

Sortie : Un ensemble C de séquences assemblées

Étape 1 : Préparation des données
- **1.1 :** Construction du k-spectre S^k et calcul des informations supplémentaires
- **1.2 :** (optionnelle) Correction des erreurs dans les reads

Étape 2 : Construction du graphe de de Bruijn $G^k(R)$
- **2.1 :** Définition de l'ensemble des nœuds
- **2.2 :** Construction des arcs
- **2.3 :** (optionnelle) Calcul de l'ensemble $\mathcal{C}^k(R)$

Étape 3 : Nettoyage du graphe de de Bruijn
- **3.1 :** (optionnelle) Correction des erreurs
- **3.2 :** (optionnelle) Résolution des répétitions

Étape 4 : Construction de l'ensemble des contigs C

Identification des contigs en tant que chemins dans $G^k(R)$

Étape 5 : (optionnelle) Scaffolding

Utilisation des reads pairés pour la résolution des répétitions et la connexion des contigs

Un nombre important d'assembleurs sont basés sur la méthode de de Bruijn comme Velvet [130], AllPaths [76], SoapDeNovo [70], Meraculos [21] ou ArpanS [102]. Utilisant une approche *parallélisée*, des assembleurs comme ABySS [111], PASHA [74], ccTSA [1] et Minia [22] distribuent l'ensemble des k-mers et leurs graphes de de Bruijn correspondants sur un réseau de processeurs. Plusieurs algorithmes utilisent le graphe de de Bruijn avec des approches alternatives. IDBA [94] utilise pour l'assemblage plusieurs graphes de de Bruijn avec des valeurs différentes de k. SparseAssembler [129] réduit la taille de l'espace mémoire utilisé pour le graphe de de Bruijn en utilisant une variante du graphe appelée *Sparse graph*. L'assembleur Cortex [49] utilise également une version de graphe de de Bruijn appelée *Colored graph* pour assembler simultanément plusieurs génomes.

Les assembleurs basés sur un graphe de de Bruijn ont l'avantage considérable de construire une vue globale du fragment d'ADN cible sans avoir besoin de calculer les chevauchements pour chaque paire de reads. En outre, en utilisant des k-mers au lieu de reads, les assembleurs de de Bruijn n'ont aucune difficulté à mélanger des reads SGS très courts avec des reads SGS plus longs. Trouver la meilleure valeur pour k reste cependant une tâche difficile. Une grande valeur pour k permet de stocker plus d'informations sur les répétitions courtes (qui restent à l'intérieur d'un même k-mer), mais perd l'information des chevauchements plus courts que la valeur $k-1$. Cela implique la nécessité d'une couverture plus élevée afin d'éviter les régions non couvertes par les reads chevauchants. Un plus petit k perd les informations sur les répétitions courtes mais capte plus d'informations sur les chevauchements entre deux reads. Le choix pour la valeur du k est donc un compromis entre la spécificité (grand k) et la sensibilité (petit k). Une solution possible est l'utilisation de différentes valeurs de k.

Les algorithmes d'assemblage de novo utilisant des données SGS ont fait beaucoup de progrès récemment. Leurs performances se rapprochent de plus en plus de celles des assembleurs utilisant les données Sanger, en matière de couverture et de précision [43, 30]. Cela a été permis en grande partie par l'approche de de Bruijn. Pourtant, l'assemblage de génomes, et surtout de grands génomes, reste un défi très important. Il y a plusieurs raisons à cela. Le principal défi du problème général d'assemblage est la détection et la résolution de répétitions. Les répétitions ont déjà été très étudiées, mais sont toujours responsables d'erreurs d'assemblage et de parties manquantes du génome cible. Un autre obstacle majeur est lié aux ressources de calcul, à savoir l'espace mémoire et le temps d'exécution. Les assembleurs de de Bruijn sont nettement meilleurs que les assembleurs Greedy et OLC concernant la durée d'exécution, mais leurs besoins en mémoire sont, en général, encore importants [132, 59]. De nouvelles méthodes proposées pour stocker le graphe de de Bruijn [129, 22] sont prometteuses en ce qui concerne les économies de mémoire.

2.1.3 Approches de l'assemblage de novo utilisant des reads PacBio

Comme mentionné précédemment (Section 1.3), la technologie PacBio produit des reads d'une longueur variable allant de 1 kbp à 20 kbp. Cette grande longueur permet aux reads de couvrir la plupart des répétitions et a le potentiel de pallier les limites des assembleurs utilisant des reads courts (SGS). Malheureusement, l'extension de la longueur des reads s'accompagne d'une augmentation significative du taux d'erreurs, qui est d'envi-

ron 16% [19]. En conséquence, la plupart des travaux de recherche se concentrant sur les reads longs ont également besoin des reads courts. Grâce à leur qualité supérieure, les reads courts sont utilisés, soit pour corriger les reads longs [4, 103, 64, 45], soit pour l'assemblage dans un contexte de novo avec des reads longs [25, 128, 6].

L'Algorithme 4 présente les lignes générales d'une approche hybride qui utilise des reads courts et des reads longs pour un assemblage de novo. On considère un ensemble SR de reads courts et un ensemble LR de reads longs sur l'alphabet $\Sigma = \{A, C, G, T\}$. Ces deux ensembles contiennent les reads obtenus après le séquençage d'un fragment d'ADN cible, ainsi que leurs compléments.

Lors de la première étape, un graphe d'assemblage $G_a(SR)$ est construit à partir de l'ensemble des reads courts SR. Cela est réalisé en assemblant les reads courts en contigs avec une approche OLC ou de de Bruijn. Chaque contig ainsi obtenu représente un nœud dans le graphe $G_a(SR)$, deux contigs étant reliés par un arc s'ils sont considérés comme l'un à côté de l'autre dans le fragment d'ADN cible.

Algorithme 4 Algorithme générique d'assemblage de novo hybride

Entrée : Un ensemble SR de reads courts et un ensemble LR de reads longs
Sortie : Un ensemble S de séquences assemblées

Étape 1 : Construction du graphe d'assemblage $G_a(SR)$
Assemblage des reads courts de SR pour obtenir l'ensemble des contigs C, chaque contig correspondant à un noeud du graphe $G_a(SR)$
Étape 2 : Alignements des contigs sur les reads longs
Calcul des alignements entre chaque contig C et chaque read long de LR
Étape 3 : Correction du graphe d'assemblage $G_a(SR)$
3.1 : Suppression des arcs non validés par les alignements des reads longs
3.2 : Résolution des répétitions
3.3 : (optionnelle) Construction des arcs entre des contigs terminaux
Étape 4 : Construction de l'ensemble des scaffolds S
Identification des scaffolds en tant que chemins dans $G_a(SR)$

Dans la deuxième étape, les alignements possibles entre les contigs et les reads longs sont calculés pour identifier les reads longs qui enjambent plusieurs contigs. Ces reads sont ensuite utilisés pour corriger le graphe d'assemblage et créer des chemins linéaires (c'est-à-dire dont les nœuds internes ont un degré entrant et sortant de 1 exactement) les plus longs possibles dans $G_a(SR)$. Cela est réalisé de la façon suivante : tout d'abord les arcs qui ne sont pas vérifiés par les alignements des reads longs sont considérés comme faux et sont donc supprimés, ensuite les contigs ayant plusieurs arcs entrants ou sortants sont considérés comme impliqués dans des répétitions. Ces contigs sont dupliqués pour créer un chemin linéaire pour chaque paire d'arc entrant et sortant dont la connexion est validée par les alignements des reads longs. Optionnellement, des arcs peuvent être rajoutés dans le graphe $G_a(SR)$ pour connecter des chemins linéaires isolés, c'est-à-dire commençant par un nœud sans arc entrant et finissant par un nœud sans arc sortant. Un arc est construit à partir du dernier nœud d'un chemin vers le premier nœud d'un autre chemin dans le cas où des reads longs s'alignent sur ces nœuds terminaux.

Parmi les algorithmes d'assemblage utilisant les deux types de reads, courts et longs, on retrouve Cerulean [25], DBG2OLC [128], AHA [6] et truSPAdes, une version de l'assembleur de novo SPAdes [5] originalement dédié uniquement aux reads courts.

En alliant la grande longueur des reads longs avec la qualité des reads courts, les assembleurs hybrides ont la capacité de construire des contigs et des scaffolds plus longs, et donc à mieux couvrir le fragment d'ADN cible. Néanmoins, dans le cas des génomes complexes avec de nombreuses répétitions, les alignements des reads longs ne suffisent pas pour corriger correctement le graphe d'assemblage, et des faux arcs peuvent y rester. De plus, le graphe d'assemblage est construit en utilisant un assembleur pour les reads courts basé sur l'approche de de Bruijn ou OLC. Comme présenté dans la Section 2.1.2, les assembleurs de reads courts sont très vulnérables aux répétitions et les contigs obtenus contiennent souvent des erreurs d'assemblage.

2.2 Détection de novo de mutations dans les génomes

Les approches détaillées dans la Section 2.1 sont dédiées à l'assemblage du fragment d'ADN cible dans sa totalité. Mais comme nous l'avons présenté dans la Section 1.2, des séquences plus réduites comme les mutations sont également analysées pour leur impact biologique. Les mutations peuvent être identifiées en comparant deux génomes, une fois que les deux ont été assemblés. Cependant, des études comme [30, 104, 132] montrent que le processus d'assemblage du génome est une tâche longue et difficile. Pour dépasser ce problème, des approches consacrées à la découverte des différentes mutations génomiques réalisent seulement un assemblage local des données SGS. Ces méthodes sont en général basées sur le graphe de de Bruijn dans lequel les mutations forment des sous-graphes avec des propriétés bien identifiées.

Ces outils sont dédiés à la détection des mutations particulières telles que les SNP avec kisSnp [95] et DISCOSNP [118] ou encore les inversions avec TAKEABREAK [68]. En utilisant la version du graphe de de Bruijn appelée *Colored graph*, des méthodes comme CORTEX_VAR [50] et Bubbleparse [67] peuvent identifier plusieurs types de mutations telles que les SNP, les insertions et les délétions.

2.3 Assemblage de novo des répétitions dans les génomes

La détection de répétitions dans le génome est importante pour étudier leurs potentielles fonctions biologiques, mais également pour améliorer la qualité de l'assemblage. Après une analyse des solutions proposées pour l'assemblage de reads SGS ou PacBio, on peut constater que les répétitions dans les génomes sont responsables de nombreuses erreurs ou manques dans l'assemblage. La procédure d'assemblage devient un problème difficile à cause de la multitude de types de répétitions, dispersées ou en tandem, exactes ou approximatives, ainsi que des longueurs des motifs et du nombre de copies variables. Dans le but d'améliorer l'assemblage des répétitions, nous nous sommes concentrés sur un type spécifique de répétitions, les répétitions en tandem. Ces travaux sont présentés dans la deuxième partie de ce mémoire.

Afin de dépasser les défis soulevés par les répétitions, plusieurs assembleurs de novo employant des données SGS ont été développés pour améliorer la détection des répétitions (voir [131, 125, 96] pour des exemples). Comme présenté dans la Section 2.1.2, les procédures dédiées à la détection des répétitions sont généralement effectuées pendant l'étape de scaffolding et se basent sur les informations des reads pairés [117]. Comme les reads d'une paire sont séparés sur le fragment d'ADN cible par une distance plus longue que la longueur des reads, les paires de reads sont utilisées pour couvrir les régions qui

contiennent de longues répétitions. Dans la recherche de méthodes pour un assemblage de novo avec des données SGS plus efficaces dans la détection de répétitions, de nouvelles structures sont également proposées comme le *graphe de de Bruijn pairé*. L'assemblage basé sur le graphe de de Bruijn pairé [82] propose l'inclusion de reads pairés directement dans la structure du graphe pour une meilleure qualité d'assemblage. En contrepartie, les problèmes deviennent algorithmiquement beaucoup plus difficiles à traiter que sur les graphes de de Bruijn classiques [57]. C'est pourquoi le graphe de de Bruijn pairé n'est pas actuellement utilisé en pratique tel quel. Son principe a dû être adapté dans les méthodes suivantes [5, 99, 122] qui sont basées sur un graphe appelé *rectangulaire*. Pour faciliter l'utilisation du graphe de de Bruijn pairé, nous avons étudiés les problèmes d'assemblage basés sur ce type de graphe afin de proposer, dans la troisième partie de ce document, des algorithmes plus rapides.

Détection des répétitions en tandem

3

Répétitions en tandem

Les répétitions en tandem sont présentes dans les génomes de la plupart des organismes et représentent l'une des caractéristiques structurales des génomes les plus étudiées [55, 79, 108, 116, 133]. Elles sont des séquences composées de copies d'un même motif situées côte à côte.

Fréquemment situées dans des gènes ou dans des régions régulatrices, les répétitions en tandem jouent un rôle important dans l'expression des gènes, l'évolution des génomes et la régulation transcriptionnelle [79, 133, 116, 120]. Elles représentent une source importante de variations du génome et sont impliquées dans des maladies génétiques héréditaires [38]. Chez l'humain, les répétitions en tandem représentent environ 3% du génome [119] et ont été associées à des maladies comme les cancers [69] et les troubles neurologiques [87].

En fonction des données disponibles, il existe plusieurs types d'approches dédiées à la détection de répétitions en tandem. Lorsque le fragment d'ADN cible est connu grâce à un assemblage préalable, la détection des répétitions en tandem peut se faire par une approche de recherche, directement dans ce fragment. Sinon, la détection doit être faite par un assemblage des reads obtenus suite au séquençage du fragment d'ADN cible. Dans ce cas, et comme dans le cas de l'assemblage global, on peut utiliser une approche mapping, en s'aidant d'une séquence de référence, ou bien une approche de novo, si l'on dispose uniquement des reads.

Ce chapitre introduit les notions utilisées dans nos travaux qui sont axés sur les répétitions en tandem. Tout d'abord, nous définissons formellement les répétitions en tandem. Ensuite, nous nous intéressons aux différentes méthodes existantes pour les détecter dans un fragment d'ADN cible connu. Ces méthodes nous permettent d'analyser la qualité de l'assemblage de novo des répétitions en tandem par les outils existants ainsi que par les algorithmes proposés dans nos travaux. Enfin, nous analysons la qualité des résultats obtenus par des assembleurs de novo, les principaux outils existants permettant un assemblage de novo des répétitions en tandem. Cette analyse nous permet de souligner le manque de méthodes adaptées à l'assemblage de novo des répétitions en tandem et permet d'établir la structure de données utilisée pour nos travaux.

3.1 Définitions

Une *répétition en tandem* (RT) est une séquence appartenant à un génome et contenant plusieurs copies (pas forcément exactes) du même motif situées les unes à côté des autres (voir Figure 3.1). Dans le cas où chaque copie est identique au motif, la séquence est appelée *répétition en tandem exacte* (RTE) ou *parfaite*. Sinon, la séquence est appelée *répétition en tandem approximative* (RTA) ou *imparfaite*. Une RT est donc définie comme suit :

Définition 3 (Répétition en tandem)**.** *Un* motif p*, appelé aussi* unité de répétition *(repeat unit) est une séquence de longueur* $|p| \geq 2$ *sur* $\Sigma = \{A, C, G, T\}$*. Une* répétition en tandem exacte *(RTE) du motif* p *est une séquence d'ADN* ε *consistant en deux ou plusieurs copies consécutives de* p*, chaque copie étant identique à* p*. Sinon, dans le cas où les copies de* p *dans* ε *ne sont pas identiques les unes aux autres, on considère une mesure de similarité* $sim(.,.)$ *et un seuil* σ*. Si* $sim(p_i, p) \geq \sigma$ *pour chaque copie* p_i *de* p *dans* ε *alors* ε *est une* répétition en tandem approximative *(RTA) du motif* p*. Une* répétition en tandem *(RT) d'un motif* p *est soit une RTE, soit une RTA du motif* p*.*

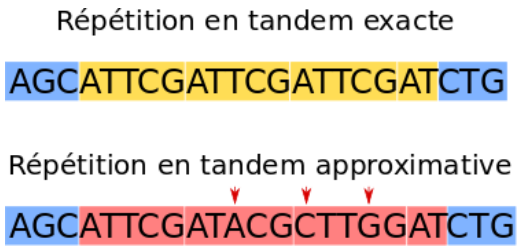


FIGURE 3.1 – Exemples de répétitions en tandem : répétition en tandem exacte (en jaune) et approximative (en rouge) du motif p=*ATTCG* avec un nombre de copies cn_p=3, 4 (dans les deux cas). Pour la répétition en tandem approximative les flèches en rouge indiquent les bases modifiées par rapport au motif p. Les séquences entourant les répétitions en tandem sont indiquées en bleu.

Soit $P = \{p_1, p_2, \ldots, p_{n_p}\}$ la liste des n_p copies de p dans une RT ε, comme dans la Définition 3. Si, pour chaque $p_i \in P$, $|p_i| = |p|$ alors la longueur de p est appelée *période* [75]. Quand p présente au moins une copie complète dans la RT ε, sa dernière copie p_{n_p} peut être partielle. Cela veut dire que p_{n_p} est en fait une copie (exacte ou approximative en fonction du type de RT de ε) d'un préfixe de p. Le *nombre de copies* cn_p de p dans ε, appelé aussi *exposant* (*exponent* en anglais) [75], est le nombre décimal $cn_p = (n_p - 1) + |p_{n_p}|/|p|$, et la RT ε peut être définie par le couple (p, cn_p).

Une RT $\varepsilon = (p, cn_p)$ est considérée comme *primitive* [115] si aucune copie de p dans ε ne contient de RT. En outre, pour identifier les RT dans un fragment d'ADN cible D, chaque RT est étendue vers la droite et vers la gauche dans D autant que possible, dans la mesure où la définition d'une RT est respectée. Pour cela, on utilise la notion de RT *maximale* et une RT $\varepsilon = (p, cn_p)$ de D est *maximale* si l'extension de ε d'un caractère à gauche ou à droite sur D ne respecte plus la définition d'une RT. Cela veut dire que la valeur de cn_p est maximum par rapport aux séquences entourant ε dans D pour que la définition d'une RT soit vérifiée pour ε.

En fonction de la longueur du motif, les RT sont classées en *microsatellites* (< 10 bp), *minisatellites* (10 à 100 bp) et *satellites* (> 100 bp). Les microsatellites sont en général des RTE [33], tandis les minisatellites et les satellites sont le plus souvent des RTA [41].

3.2 Détection des répétitions en tandem dans un fragment connu

Le problème de la détection des RT a fait l'objet de nombreuses études en raison du rôle biologique important des RT, ainsi que pour améliorer la qualité d'assemblage [86]. La détection de RT peut se faire par une recherche directe dans un fragment d'ADN connu (génome, contig ou même read). Lorsque la succession des nucléotides dans le fragment d'ADN cible n'est pas connue, la détection des RT peut se faire par l'assemblage global du fragment d'ADN cible ou par un assemblage local, en ne reconstituant que les régions associées aux RT.

Dans cette section nous présentons la définition du problème de recherche des RT dans des fragments d'ADN connus ainsi que les différentes solutions existantes à ce problème.

3.2.1 Description du problème de recherche des répétitions en tandem dans un fragment d'ADN connu

Lorsque le fragment d'ADN cible est connu (*i.e.* la succession des nucléotides a été déterminée), l'identification des RT peut être réalisée par une recherche directe dans ce fragment.

Dans sa forme la plus simple, une RT est appelée *répétition en tandem carrée*. Elle est primitive et constituée de deux copies consécutives, complètes et exactes d'un motif p. Le problème de recherche des RT carrées découle alors d'un problème classique, le PROBLÈME DE LA PLUS LONGUE SOUS-CHAÎNE COMMUNE [44].

PROBLÈME DE LA PLUS LONGUE SOUS-CHAÎNE COMMUNE

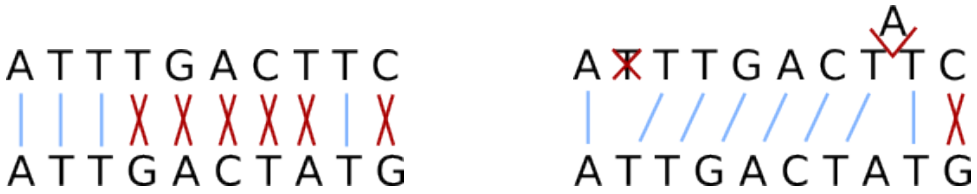
Entrée : Un fragment d'ADN cible D et x, y deux positions dans D.

Objectif : Trouver la plus longue sous-chaîne p de D telle que $p = D[x, |p|] = D[y, |p|]$.

Ce problème peut être résolu en $O(n)$ pour $n = |D|$ par une comparaison itérative pour chaque $i \geq 0$ des caractères $D[x + i]$ et $D[y + i]$. Pour la recherche d'une RT carrée, il suffit donc de rajouter la condition $y = x + |p|$. Pour une recherche efficace des RT carrées, il faut considérer toutes les positions possibles des RT carrées dans le fragment D. Dans un fragment D, il a été démontré qu'il existe au maximum $O(n \log n)$ RT carrées pour $n = |D|$ [61]. Pour étendre la recherche aux RT exactes et maximales, ces dernières sont considérées comme des séries maximales des RT carrées d'un même motif p avec des positions contiguës des départs (*i.e.* x, $x + p$, $x + 2p$, etc). En utilisant cet encodage, il a été prouvé qu'il y a au plus $O(n)$ RT exactes et maximales dans un fragment D de longueur $|D| = n$ et qu'elles peuvent être trouvées en $O(n)$ [61].

Comme mentionné dans la Définition 3, pour une RT approximative ε d'un motif p, les copies de p dans ε ne sont pas identiques entre elles. Pour étendre la recherche des RT aux RT approximatives, il faut donc introduire une mesure de similarité ou une distance pour limiter le nombre autorisé de différences entre les copies. Ces deux types de calculs sont complémentaires. Pour une mesure de similarité, on attribue des points aux positions similaires, alors que pour calculer une distance, on attribue des points aux différences, appelées également mutations. Ces mutations sont de trois types : les substitutions, les délétions et les insertions. En fonction des types d'erreurs prises en compte, il existe plusieurs calculs possibles. Les plus classiques sont la *distance de Hamming* qui n'autorise

que des substitutions (Figure 3.2 (a)), et la *distance de Levenshtein* qui permet les insertions et délétions en plus des substitutions (Figure 3.2 (b)).



(a) Exemple de distance de Hamming (b) Exemple de distance de Levenshtein

FIGURE 3.2 – Exemples de distances pour deux séquences *ATTGACTTC* et *ATTGACTAG*. En fonction des erreurs prises en compte, la distance est égale à 6 si l'on n'autorise que les substitutions (a) ou à 3 si l'on autorise aussi les insertions et les délétions (b). De façon équivalente, le nombre de positions identiques est différent : 4 si l'on n'autorise que les substitutions (a) et 8 si l'on autorise aussi les insertions et les délétions (b)

Définition 4 (Distance de Hamming)**.** *Soient r et s deux séquences de longueur $|r| = |s| = n$ sur l'alphabet $\Sigma = \{A, C, G, T\}$. La distance de Hamming $dist_H(r, s)$ entre r et s est égale au nombre de substitutions de caractères nécessaires pour transformer r en s.*

Définition 5 (Distance de Levenshtein)**.** *Soient r et s deux séquences de longueur $|r| = n$ et $|s| = m$ sur l'alphabet $\Sigma = \{A, C, G, T\}$. La distance de Levenshtein $dist_L(r, s)$ entre r et s est égale au nombre minimum de substitutions, insertions et délétions de caractères nécessaires pour transformer r en s.*

Dans le cas d'une mesure de similarité entre les copies d'un motif p, les positions similaires sont *alignées* et le *score d'alignement* est calculé en attribuant des poids aux position similaires. Il est possible également d'attribuer des poids aux insertions, délétions ou substitutions.

Définition 6 (Alignement)**.** *Soient r et s deux séquences de longueur $|r| = n$ et $|s| = m$ sur l'alphabet $\Sigma = \{A, C, G, T\}$. L'alignement $align(r, s)$ de r et s est représenté par le couple (r^*, s^*) où r^* et s^* sont deux séquences sur l'alphabet $\Sigma^* = \{A, C, G, T, -\}$ tels que :*

- $|r^*|$=$|s^*|$,
- *soit $r^*[i] \neq$ " $-$ " soit $s^*[i] \neq$ " $-$ " pour tout $1 \leq i \leq |r^*|$,*
- *en supprimant tous les caractères " $-$ " de r^* et de s^* on obtient respectivement r et s.*

Définition 7 (Score d'un alignement)**.** *Le score $sA(r^*, s^*)$ d'un alignement $align(r, s) = (r^*, s^*)$ est égal au $sA(r^*, s^*) = \sum_{i=1}^{|r^*|} w(r^*[i], s^*[i])$ où $w(\cdot, \cdot)$ est une fonction de poids pour chaque type d'opération :*

- *insertion ($s^*[i] =' -'$ et $r^*[i] \in \Sigma$)*
- *délétion ($r^*[i] =' -'$ et $s^*[i] \in \Sigma$)*
- *substitution ($r^*[i] \neq s^*[i]$ et $r^*[i], s^*[i] \in \Sigma$)*

- *équivalence ($r^*[i] = s^*[i]$ et $r^*[i], s^*[i] \in \Sigma$).*

Une fois le choix pour la mesure de similarité (ou la distance) fait, la recherche des répétitions en tandem peut être définie de la façon suivante :

PROBLÈME DE LA RECHERCHE DES RÉPÉTITIONS EN TANDEM

Entrée : Un fragment d'ADN cible D , une mesure de similarité $sim(.,.)$ et un seuil σ.

Objectif : Trouver toutes les RT maximales définies par des (n_p+1)-uplets $(x, p_1, p_2, \ldots, p_{n_p})$ où x représente la position de début dans D d'une RT ayant n_p copies d'un motif p sur $\Sigma = \{A, C, G, T\}$ tels que $p_i = D[x + \sum_{1 \leq j < i} |p_j|, |p_i|]$ et $sim(p_i, p) \geq \sigma$ pour tout $i, j \leq n_p$.

Pour le PROBLÈME DE LA RECHERCHE DES RÉPÉTITIONS EN TANDEM, il existe plusieurs variantes en fonction de la définition des RTA. Par exemple, il est possible de définir les RTA avec la condition $sim(p_i, p_{i+1}) \geq \sigma$ pour tout $i < n_p$ ou avec la condition $sim(p_i, p_j) \geq \sigma$ pour tout $i \leq n_p$ et tout $1 \leq j < i$. Il a été prouvé que dans le cas d'une distance de Hamming ou d'une mesure de similarité qui ne considère que les substitutions, l'algorithme le plus efficace pour la recherche de RT a une complexité en $O(nk \log(k) + S)$ où k est le nombre maximum de substitutions possibles et S le nombre de RT trouvées [62].

3.2.2 Approches de recherche des répétitions en tandem dans des fragments d'ADN connus

La détection des RT dans des séquences de référence de génomes, de chromosomes ou d'autres types de fragments d'ADN assemblés peut être effectuée par un nombre important d'outils existants. Les différences dans les approches algorithmiques et les résultats de ces outils ont été présentés dans plusieurs études [108, 66, 83, 84, 71].

L'Algorithme 5 présente les lignes générales des méthodes utilisées pour la détection des RT dans des fragments d'ADN connus. L'algorithme contient deux étapes principales : la détection des RT et le filtrage des RT [83]. L'étape de détection des RT est le noyau de l'algorithme de recherche et résout le PROBLÈME DE LA RECHERCHE DES RÉPÉTITIONS EN TANDEM tel qu'il a été défini plus haut. L'efficacité globale du programme est déterminée par l'algorithme utilisé à cette étape. Basé sur certains critères de sélection, l'algorithme détecte les RT spécifiées par les paramètres d'entrée des utilisateurs (longueur du motif ou de la RT, pourcentage de similarité entre les copies etc.).

Une étape de filtrage est ensuite rajoutée pour identifier des répétitions biologiquement significatives et pour éliminer les différents types de redondance. Optionnellement, les outils de recherche des RT peuvent donner des informations détaillées sur chaque RT comme les séquences entourant les RT, des statistiques globales sur le fragment d'ADN cible ou même des informations supplémentaires pour une analyse ultérieure. Les deux étapes de l'algorithme de recherche des RT sont détaillées par la suite.

Détection des RT : La recherche peut être réalisée pour des motifs spécifiques. Dans ce cas, l'utilisateur peut fournir une liste de motifs qui seront recherchés dans tout le fragment d'ADN cible [18, 56]. Cependant, et de manière générale, les motifs ou les RT ne sont pas connus et la recherche de RT dépend uniquement de la nature répétitive de

Algorithme 5 Algorithme générique de détection des répétitions en tandem

Entrée : Un fragment d'ADN D
Sortie : Un ensemble $\mathcal{RT}$ de répétitions en tandem

Étape 1 : Détection
Détection des RT dans le fragment D
Étape 2 : Filtrage
2.1 : Recherche des RT chevauchantes
2.2 : Choix du motif pour les RT chevauchantes
2.3 : Élimination des RT redondantes

parties du fragment d'ADN cible. Dans ce cas, les RT potentielles sont recherchées, soit de façon exhaustive, soit avec une heuristique ou une méthode statistique.

Pour une approche exhaustive, les outils implémentent souvent un algorithme combinatoire qui découpe tout le fragment d'ADN cible D en sous-séquences. Ensuite, les RT sont identifiées à partir de chaque position dans D par comparaison des sous-séquences adjacentes. La complexité de ces algorithmes croît exponentiellement, surtout quand la recherche inclut des RTA [27]. Plusieurs outils réalisent une recherche exhaustive comme mreps [60], IMEx [88] ou autres [65, 113].

Les approches heuristiques ou statistiques utilisent des algorithmes qui analysent tout d'abord les séquences génomiques pour détecter les régions qui peuvent être des RT en suivant des règles statistiques données. Pour cela, elles utilisent des petites séquences (windows) pour parcourir le fragment d'ADN cible. En partant de ces séquences, les méthodes détectent d'abord les régions répétitives (en général des petites répétitions exactes qui ne sont pas nécessairement en tandem). Ces régions sont ensuite soumises à des tests de validation et associées pour déterminer les RT plus longues. Cet ensemble de RT peut ne pas être exhaustif car certaines sous-séquences, qui pourraient passer des tests de validation, peuvent ne pas être détectées par des tests statistiques. La complexité de ces algorithmes est cependant plus réduite [27] mais il est difficile de garantir l'exhaustivité de la recherche qui dépend du choix de la taille des séquences de départ [83]. Parmi les outils basés sur cette approche, on retrouve Tandem repeats finder [7], T-REKS [54], TRStalker [92] ou autres [10, 126].

La complexité de cette étape dépend également du type de RT recherchées par les outils. Certaines de ces méthodes de recherche sont spécialement conçues pour les RTE comme BWtrs [97], tandis que d'autres peuvent également détecter les RTA, soit en ne prenant en compte que les substitutions comme mreps [60], soit en considérant tous les types de mutations (substitutions, insertions et délétions) comme TRF [7]. Par la suite nous allons détailler les algorithmes utilisés pour la détection des RT de deux outils : BWtrs [97] dans le cas de la recherche de RTE et de mreps [60] pour la recherche de RTA.

La plupart des algorithmes de recherche exacte de motifs dans une séquence utilisent une structure de données appelée le *tableau des suffixes* [77].

Définition 8 (Tableau de suffixes [77])**.** *Soit une séquence s de longueur $|s| = n$. Le tableau des suffixes $T(s)$ de s est un tableau d'entiers avec $n+1$ cases contenant les $n+1$ suffixes de la séquence $s+\$$ en ordre lexicographique.*

En indexant tous les suffixes d'une séquence s, le tableau des suffixes permet de localiser rapidement toutes les occurrences d'un motif p dans s. Cela est fait en recherchant

tous les suffixes pour lesquels p est un préfixe. L'algorithme de détection des RTE implémenté par BWtrs [97] utilise une structure connexe au tableau des suffixes, la *transformée de Burrows-Wheeler* [13].

Définition 9 (Transformée de Burrows-Wheeler [13])**.** *Soient une séquence s de longueur $|s| = n$ et $T(s)$ le tableau des suffixes de s. La transformée de Burrows-Wheeler est un tableau défini de la façon suivante :*

$$BWT(s)[i] = \begin{cases} s[T(s)[i]-1] & \textit{si } T(s)[i] > 0 \\ \$ & \textit{si } T(s)[i] = 0 \end{cases}$$

La transformée de Burrows-Wheeler représente une façon réversible de permuter les caractères de s pour former le tableau $BWT(s)$, la séquence s pouvant être reconstruite à partir de $BWT(s)$. L'avantage de cette transformée par rapport au tableau des suffixes est qu'elle a tendance à regrouper des occurrences d'un même caractère et donc à être plus facile à compresser [13]. La recherche des occurrences d'un motif p dans la séquence s peut se faire avec la transformée de Burrows-Wheeler de façon linéaire par rapport à la longueur de p, c'est-à-dire en $O(|p|)$ indépendamment de la longueur de s [34]. Néanmoins, cette recherche utilise un tableau $\mathcal{O}(s)$ préalablement calculé qui contient $|\Sigma|$ cases où Σ représente l'alphabet de s. Pour chaque caractère $c \in \Sigma$, $\mathcal{O}(s)[c]$ contient la somme du nombre d'occurrences dans la séquence s des caractères plus petits que c dans Σ en ordre lexicographique. Par exemple, si $\Sigma = \{A, C, G, T\}$ et $s = ACCTCAG$ alors $\mathcal{O}(s)['A'] = 0$, $\mathcal{O}(s)['C'] = 2$,$\mathcal{O}(s)['G'] = 5$ et $\mathcal{O}(s)['T'] = 6$. L'Algorithme 6 [34] détaille les étapes de la recherche d'un motif p dans une séquence s avec la transformée de Burrows-Wheeler. À chaque étape i dans l'Algorithme 6, les indices deb et fin indiquent la position dans le tableau $T(s)$ des suffixes respectivement du premier et du dernier suffixes en ordre lexicographique ayant comme préfixe $Suff(p, i)$. La fonction $Occ(c, pos)$ renvoie le nombre d'occurrences d'un caractère c dans $BWT(s)[1, pos]$. Cette fonction peut être calculée en $O(1)$ [34].

Algorithme 6 Algorithme de recherche de motifs basé sur BWT : RechercheBWT [34]

Entrée : Un motif p et un tableau $\mathcal{O}(s)$ calculé pour une séquence s sur un alphabet Σ tel que $\mathcal{O}(s)$ contient pour chaque caractère $c \in \Sigma$ la somme du nombre d'occurrences dans la séquence s des caractères plus petits que c dans Σ en ordre lexicographique.

Sortie : Un intervalle $[deb, fin]$ représentant les indices dans le tableau $T(s)$ des suffixes contenant p comme préfixe

$c \leftarrow p[|p|]$
$i \leftarrow |p|$
$deb \leftarrow \mathcal{O}[c] + 1$
$fin \leftarrow \mathcal{O}$[le caractère suivant $c \in \Sigma$ en ordre lexicographique]
Tant que $deb \leq fin$ **et** $i \geq 2$ **faire**
  $c \leftarrow p[i-1]$
  $deb \leftarrow \mathcal{O}[c] + Occ(c, deb-1) + 1$
  $fin \leftarrow \mathcal{O}[c] + Occ(c, fin)$
  $i \leftarrow i - 1$
fin Tant que
retourner $[deb, fin]$

L'Algorithme 6 [34] est ensuite utilisé par BWTrs afin de rechercher pour chaque motif p les occurrences des motifs $p+p+c$ où $c \in \Sigma$ et $c \neq p[1]$. Les étapes de la recherche

de RTE implémentée par BWtrs sont présentées dans l'Algorithme 7. Le principe de base de cet algorithme est de rechercher les RTE de chaque motif p possible.

Pour cela, BWtrs commence avec le motif vide p auquel la méthode $Etendre(p, \Sigma)$ ajoute de manière récursive les caractères de l'alphabet Σ. Pour chaque motif, BWtrs calcule l'intervalle $[k, l]$ qui indique les positions des suffixes ayant comme préfixe p dans le tableau des suffixes $T(D)$ du fragment d'ADN D. Cela est réalisé avec l'Algorithme 6. De la même manière, BWtrs calcule l'intervalle $[i, j]$ pour chaque $p + p + c$ où $c \in \Sigma$ et $c \neq p[1]$. Ensuite l'existence d'une RTE $p + p$ maximale à droite (c'est-à-dire qui ne peut pas être étendue à droite) est vérifiée en comparant les valeurs calculées de $[i, j]$ et $[k, l]$. La RTE $p + p$ est ensuite étendue à gauche pour retrouver la RTE maximale.

Algorithme 7 Algorithme de BWtrs pour la détection des RTE [97]

Entrée : Un fragment d'ADN D
Sortie : Un ensemble $\mathcal{RTE}$ de répétitions en tandem exactes

$p \leftarrow''$
$p \leftarrow Etendre(p, \Sigma)$
Pour chaque motif p **faire**
  $[k, l] \leftarrow RechercheBWT(p, \mathcal{O}(D))$
  Pour chaque caractère $c \in \{A, C, G, T\} - p[1]$ **faire**
    $[i, j] \leftarrow RechercheBWT(p + p + c, \mathcal{O}(D))$
    Si $k \leq i \leq j \leq l$ **alors**
      Pour chaque occurrence de $\beta = p + p + c$ dans D **faire**
        Extension répétée à gauche de β
        $\mathcal{RTE} = \mathcal{RTE} \cup \beta$
      fin Pour
    fin Si
  fin Pour
  $p \leftarrow Etendre(p, \Sigma)$
fin Pour
retourner $\mathcal{RTE}$

Comme précisé auparavant, dans le cas de la détection des RTA, les algorithmes utilisent des définitions différentes pour les RTA en fonction des types de mutations pris en compte. L'algorithme implémenté par mreps [60] prend en compte uniquement les substitutions en utilisant les notions de *k-répétition* et *k-run* basées sur la distance de Hamming $dist_H(\cdot, \cdot)$.

Définition 10 (k-répétition [62])**.** *Une séquence s de longueur $|s| = n$ est une k-répétition avec la période ϕ, où $\phi \leq n/2$ si $dist_H(s[1, n - \phi], s[p + 1, n - \phi]) \leq k$.*

Définition 11 (k-run [62])**.** *Une séquence s de longueur $|s| = n$ est un k-run avec la période ϕ, où $\phi \leq n/2$ si pour chaque $1 \leq i \leq n - 2\phi + 1$, $dist_H(s[i, \phi - 1], s[i + \phi, \phi - 1]) \leq k$, c'est-à-dire la sous-séquence $s[i, 2\phi - 1]$ est une k-répétition.*

De façon équivalente, une séquence s de longueur $|s| = n$ est une k-répétition avec la période ϕ si le nombre de positions i tel que $s[i] \neq s[i + \phi]$ est au plus k. Un k-run peut être vu comme une séquence de RTA carrées, c'est-à-dire avec 2 copies c_1 et c_2 telles que $|c_1| = |c_2| = \phi$. De plus, chaque k-répétition fait partie d'un k-run avec la même période et chaque k-run est l'union des toutes les k-répétitions qu'il contient. De façon similaire

à un RT maximal, un k-run dans une séquence s est *maximal* s'il ne peut pas être étendu à gauche ou à droite tout en respectant sa définition. L'objectif de mreps est de rechercher des k-run maximaux (voir Algorithme 9 [60, 62]). Pour cela, il divise le fragment d'ADN en des séquences plus petites appelées *facteurs* avec la *factorisation de Lempel-Ziv* [62].

Définition 12 (Factorisation de Lempel-Ziv [62])**.** *La Factorisation de Lempel-Ziv d'une séquence s est $s = f_1 + f_2 + \ldots + f_m$ où chaque facteur f_i pour $1 \leq i \leq m$ est défini de la façon suivante :*

- $f_1 = s[1]$
- *pour $i \geq 2$, f_i est la plus courte sous-séquence de s qui n'apparaît pas dans $f_1 + f_2 + \ldots + f_{i-1}$.*

Ensuite, les facteurs sont regroupés dans des *blocs* et mreps recherche des k-runs contenant la fin d'un bloc ou d'un facteur. Cela est réalisé en utilisant les notions qui suivent. Soit une séquence s de longueur $|s| = n$ et une répétition $r = s[i, j]$ dans s avec une période ϕ. La sous-séquence $s[i, \phi - 1]$ est appelée *racine de gauche* de r et la sous-séquence $s[j + \phi - 1, \phi - 1]$ représente la *racine de droite* de r [62]. La répétition r *touche* $s[pos]$ si r contient le caractère de s situé à la position $pos - 1$, pos ou $pos + 1$ [62].

Algorithme 8 Algorithme de détection des K-run situés à gauche de la position pos dans D [62]

Entrée : Un fragment d'ADN D de longueur n et trois entiers : la période minimale ϕ_0, le nombre maximal de substitutions K et la position pos, $1 \leq pos \leq |D|$

Sortie : Les K-runs avec une période minimum ϕ_0 et situés à gauche de la position pos dans D

Pour ϕ=ϕ_0 **à** $pos - 1$ **faire**
  Pour k=0 **à** K **faire**
    Calcul de $LP_k(\phi, pos)$ et $LS_k(\phi, pos)$
  fin Pour
fin Pour
Pour ϕ=ϕ_0 **à** $min(n - pos + 1, n/2)$ **faire**
  Pour k=0 **à** K **faire**
    Si $LP_k(\phi, pos) + LS_{K-k}(\phi, pos) \geq \phi$ et $LP_k(\phi) \leq \phi$ **alors**
      Création d'un k-run commençant à la position $max(pos + \phi - LS_{K-k}(\phi) - 1, pos - 1)$ et finissant à la position $min(pos + LP_k(\phi) - 1, pos + \phi - 1)$
    fin Si
  fin Pour
fin Pour

La recherche des k-runs contenant la dernière position pos d'un bloc ou d'un facteur a deux étapes. Tout d'abord, mreps recherche les k-runs à gauche de pos (voir Algorithme 8), c'est-à-dire les k-runs qui ont leur racine de droite commençant à une position x telle que $x < pos$. Ensuite, de façon similaire, l'algorithme de mreps recherche les k-runs à droite de pos, c'est-à-dire les k-runs qui ont leur racine de droite commençant à une position x telle que $x > pos$. Comme BWtrs, mreps utilise une méthode de compression du texte, la technique des fonctions de la plus longue extension [62] :

$$LP_k(\phi, l) = max\{j | dist_H(s[l-\phi, j-1], s[l, j-1]) \leq k\}$$
$$LS_k(\phi, l) = max\{j | dist_H(s[l-\phi-j, j-1], s[l-j, j-1]) \leq k\}$$

À la fin de chaque étape, les k-runs détectés qui se chevauchent sont fusionnés pour former des k-run maximaux.

Algorithme 9 Algorithme de mreps pour la détection des RT [60, 62]

Entrée : Un fragment d'ADN D, un entier K
Sortie : Les runs maximaux de RT avec K substitutions

Étape 1 : Préparation du fragment D
1.1 : Calcul de la factorisation Lempel-Ziv $D = f_1 \dots f_m$
1.2 : Division de la factorisation dans des blocs de $K + 2$ facteurs consécutifs $D = B_1 \dots B_q$
Étape 2 : Recherche des K-runs qui *touchent* le dernier caractère d'un bloc

Pour chaque bloc B_i **faire**
2.1 : Recherche des k-runs qui *touchent* le dernier caractère de B_i, pour k de 0 à K
2.2 : Fusion des k-runs chevauchants
fin Pour

Étape 3 : Recherche des K-runs qui *touchent* le dernier caractère d'un facteur qui n'est pas le dernier dans un bloc

Pour chaque bloc B_i **faire**
3.1 : Division de $B_i = f_j \dots f_{j+o}$ en deux sub-blocs : $B_i = B'_i B''_i$ tels que $B'_i = f_j \dots f_{\lfloor o/2 \rfloor}$ et $B''_i = f_{\lfloor o/2 \rfloor + 1} \dots f_{j+o}$
3.2 : Recherche des k-runs qui *touchent* le dernier caractère de B'_i, pour k de 0 à K
3.3 : Fusion des k-runs chevauchants
3.4 : Répéter récursivement les étapes **3.1, 3.2** et **3.3** pour B'_i et B''_i jusqu'à ce que les blocs contiennent un seul facteur
fin Pour

Étape 4 : Recherche des K-runs dans chaque facteur de D

Pour chaque facteur f_i **faire**
2.1 : Recherche des k-runs qui se trouvent entièrement dans le facteur f_i, pour k de 0 à K
2.2 : Fusion des k-runs chevauchants
fin Pour

Filtrage des RT : Dans la deuxième phase, diverses méthodes de filtrage sont utilisées pour identifier et extraire les RT biologiquement significatives. L'étape de détection des RT peut introduire de la redondance dans l'ensemble des RT identifiées. Dans de nombreux cas, les méthodes utilisées pendant la première étape peuvent détecter une même RT plusieurs fois. C'est le cas lorsqu'une RT peut être représentée avec deux motifs différents ou lorsque deux séquences répétées en tandem se chevauchent. La redondance n'a

pas de sens biologique et résulte essentiellement des méthodes mises en œuvre par les algorithmes. Cependant, d'un point de vue biologique, une base donnée dans un fragment d'ADN cible appartient à une seule RT [83].

Ainsi, des outils comme INVERTER [127] préfèrent signaler toutes les RT qui se chevauchent et laisser l'utilisateur faire le filtrage. D'autres comme Tandem repeats finder [7] ou mreps [60] renvoient seulement les RT considérées comme les plus probables parmi celles qui se chevauchent. Pour cela, les outils calculent le motif qui minimise le taux d'erreur moyen des RT chevauchantes. Ces différents choix provoquent de grandes disparités dans l'ensemble des RT détectées par les différents outils.

3.3 Évaluation de la détection de novo des répétitions en tandem

Dans le cas d'un contexte de novo, la détection des RT reste un problème difficile [117]. Les principaux outils qui permettent la résolution des RT à partir des reads et sans utiliser une séquence de référence sont les assembleurs de novo. L'objectif principal de ces outils est de produire des séquences assemblées les plus longues possible et de haute qualité. Comme nous l'avons présenté dans le deuxième chapitre (Section 2.1.2), pour atteindre cet objectif, les assembleurs utilisent des heuristiques qui peuvent laisser de nombreuses répétitions, en particulier des RT, non résolues en raison de la complexité des génomes [117].

Afin d'analyser la qualité de la résolution des RT par les assembleurs existants, nous avons choisi quatre assembleurs parmi les meilleurs assembleurs de données SGS autant par rapport à la qualité de l'assemblage que par rapport au temps d'exécution et à l'espace mémoire nécessaires [104, 132, 30]. Ces quatre assembleurs sont basés sur les trois principales méthodes d'assemblage : SSAKE [123] utilisant l'approche Greedy, SGA [110] basé sur l'approche OLC, ainsi que ABySS [111] et Velvet [130] utilisant l'approche de de Bruijn.

Pour analyser le comportement des assembleurs face aux RT, nous avons utilisé deux organismes différents : le nématode *Caenorhabditis elegans* et la bactérie *Legionella pneumophila*. Le génome de *C. elegans* est très complexe. En effet, les assembleurs ont besoin d'un temps de calcul élevé pour obtenir un faible pourcentage de contigs correctement mappés sur sa séquence [132]. Ceci est principalement dû au nombre important de répétitions, et en particulier des RT. Pour nos expériences, nous avons utilisé le premier chromosome de *C. elegans*. Sa séquence, dont la longueur est environ de 15Mo, a été téléchargée à partir de GenBank [GenBank : GCA_000002985.3]. Comme détaillé par la suite dans la Section 3.3.2, ce chromosome possède une structure très complexe et contient plus de RT que de nombreux génomes complets. Pour le deuxième organisme choisi, *L. pneumophila*, nous avons utilisé le génome de la souche appelée *Philadelphia* pour lequel des RT ont été étudiées afin d'identifier leur rôles biologiques [23, 121]. La séquence du génome a une longueur d'environ 3,4 Mb [GenBank : GCA_000008485.1]. Comme présenté dans le Tableau 3.1, nous avons utilisé des reads pairés obtenus avec la technologie Illumina pour *L. pneumophila* et des reads pairés simulés (modèle d'erreurs d'Illumina) pour le premier chromosome de *C. elegans*.

Pour chaque ensemble de reads, nous avons exécuté chaque assembleur en faisant varier ses paramètres principaux : la longueur minimum pour qu'un chevauchement entre deux reads soit pris en compte pour SSAKE et SGA et la longueur des k-mers pour Velvet

TABLEAU 3.1 – Données utilisées

Organisme	**C. elegans** Chromosome I	**L. pneumophila** Génome de la souche Philadelphia
Taille séquence de référence	15.072.423 bp	3.397.754 bp
Longueur reads	100 bp	100 bp
Type de reads	Simulés avec GemSim [80]	Réels [SRA : SRX258262]
Couverture	20x et 100x	190x
Taille d'insert	600 bp	400 bp

et ABySS. Les résultats obtenus ont tout d'abord été analysés du point de vue global de l'assemblage afin d'étudier l'impact des RT sur la qualité des contigs et des scafflods. Ensuite, nous avons effectué une analyse de la qualité de résolution des RT.

3.3.1 Qualité globale de l'assemblage

Chacun des quatre assembleurs choisis suit les principales étapes présentées dans le chapitre précédent (Section 2.1.2) : la correction des reads pour l'obtention des contigs et l'utilisation de l'information fournie par les paires de reads pour construire les scaffolds. Néanmoins, ABySS [111] est le seul assembleur parmi les quatre choisis qui retourne à la fin d'une exécution l'ensemble des contigs ainsi que l'ensemble des scaffolds, les trois autres retournant uniquement l'ensemble des scaffolds. En conséquence, la qualité de l'assemblage a été analysée sur l'ensemble des scaffolds pour les quatre assembleurs ainsi que sur celui des contigs obtenus pour ABySS. Pour cela, nous utilisons des métriques simples comme le nombre de séquences dans chaque ensemble et leur longueur totale.

Nous avons également calculé des métriques plus complexes utilisant les méthodes employées par GAGE [104]. La première est le *N50* qui représente la longueur du plus petit scaffold (ou contig) telle que 50 % de la séquence de référence est contenue dans les scaffolds (ou contigs) qui ont au moins cette longueur. La deuxième est le pourcentage de bases erronées qui représente le pourcentage des bases des scaffolds (ou contigs) que GAGE n'aligne pas correctement sur la séquence de référence. Enfin, la troisième métrique est le pourcentage de bases manquantes qui évalue le pourcentage des bases de la séquence de référence qui n'ont pu être alignées par GAGE sur aucun scaffold (ou contig).

Les résultats obtenus avec SSAKE sont détaillés dans le Tableau 3.2. Nous pouvons observer que la qualité des séquences assemblées est très basse sur les deux organismes testés. SSAKE est incapable d'étendre correctement les reads à cause des multiples répétitions présentes dans les deux séquences de référence. De plus, la variation de la longueur minimum du chevauchement n'a pas d'influence significative sur la qualité ou le nombre de scaffolds obtenus. Après chaque exécution, plus de 80% des reads restent non-assemblés et plus de 95% des séquences assemblées sont erronées.

En analysant les résultats obtenus avec SGA, on observe une amélioration de leur qualité par rapport à ceux obtenus avec l'assembleur Greedy (voir Tableau 3.3). Moins de 0,2% des bases des séquences assemblées par SGA n'ont pas été alignées sur la séquence de référence par GAGE. Cependant, le niveau de recouvrement de la séquence de réfé-

TABLEAU 3.2 – Qualité globale de l'assemblage avec SSAKE

Longueur minimum du chevauchement		**17 bp**	**27 bp**	**37 bp**	**47 bp**
Scaffolds C. elegans chromosome I	**Nombre de séquences**	2.164	2.163	1.940	774
	N50	-	-	-	-
	Longueur totale	47 kb	46,6 kb	41,5 kb	16,7 kb
	Pourcentage de bases erronées	95,71 %	96,96 %	97,72 %	96,67 %
	Pourcentage de bases manquantes	99,76 %	99,79 %	99,81 %	99,80 %
Scaffolds L. pneumophila souche Philadelphia	**Nombre de séquences**	181	181	154	15
	N50	-	-	-	-
	Longueur totale	3,9 kb	3,9 kb	3,3 kb	315 bp
	Pourcentage de bases erronées	97,00 %	97,00 %	100 %	100 %
	Pourcentage de bases manquantes	99,98 %	99,98 %	100 %	100 %

rence est très différent entre les deux organismes testés. Le premier chromosome de *C. elegans* ayant un nombre significatif de répétitions, le graphe de chevauchement possède une structure très complexe. L'assembleur SGA ne peut pas linéariser de nombreux chemins, il décide donc de laisser ces parties du graphe de chevauchement non-assemblées. Comme pour SSAKE, la variation de la longueur minimum de chevauchement entre les reads a très peu d'impact sur la qualité des résultats.

TABLEAU 3.3 – Qualité globale de l'assemblage avec SGA

Longueur minimum du chevauchement		**17 bp**	**27 bp**	**37 bp**	**47 bp**
Scaffolds C. elegans chromosome I	**Nombre de séquences**	14.425	8.165	5.690	4.360
	N50	-	-	-	-
	Longueur totale	2,4 Mb	1,8 Mb	1,3 Mb	1 Mb
	Pourcentage de bases erronées	0,12%	0,05%	0,01 %	0,01 %
	Pourcentage de bases manquantes	86,70 %	87,99 %	89,95 %	91,93 %
Scaffolds L. pneumophila souche Philadelphia	**Nombre de séquences**	1.217	370	331	326
	N50	58.687	119.899	150.373	150.373
	Longueur totale	3,5 Mb	3,4 Mb	3,4 Mb	3,4 Mb
	Pourcentage de bases erronées	0,08 %	0,07 %	0,07 %	0,07 %
	Pourcentage de bases manquantes	0,01 %	0,03 %	0,03 %	0,04 %

Les résultats obtenus avec Velvet sont présentés dans le Tableau 3.4. Une fois de plus, nous pouvons observer une amélioration de la qualité des séquences assemblées par rapport à celles obtenues avec SSAKE mais également par rapport à celles obtenues avec SGA. Cela est dû à la structure du graphe de de Bruijn qui, en découpant les reads en k-mers, permet une meilleure résolution des répétitions. En augmentant la longueur des k-mers, nous obtenons des scaffolds de meilleure qualité, aussi bien par rapport au pourcentage d'erreurs réalisées pendant l'assemblage mais également par rapport à la couverture de la séquence de référence. Cela est valable pour les deux organismes testés. Comme précisé auparavant, une grande valeur pour k permet de stocker plus d'informations sur les répétitions courtes, c'est-à-dire d'une longueur inférieure à k. Les répétitions courtes étant très nombreuses dans les deux séquences de référence, plus les k-mers sont petits, plus ils se chevauchent. En conséquence, le degré moyen des nœuds du graphe de de Bruijn décroît en augmentant la longueur des k-mers et le graphe est plus rapide à assembler.

TABLEAU 3.4 – Qualité globale de l'assemblage avec Velvet

Longueur k-mers		**17 bp**	**27 bp**	**37 bp**	**47 bp**
Scaffolds C. elegans chromosome I	**Nombre de séquences**	393.442	78.725	27.758	23.925
	N50	37	874	2.746	1.954
	Longueur totale	15,6 Mb	16,6 Mb	15,3 Mb	15,5 Mb
	Pourcentage de bases erronées	72,21 %	17,74 %	0,44 %	0,05 %
	Pourcentage de bases manquantes	70,89 %	9,57 %	1.46 %	0,75 %
Scaffolds L. pneumophila souche Philadelphia	**Nombre de séquences**	185.114	77.927	26.113	5.638
	N50	55	146	415	1.553
	Longueur totale	7,9 Mb	6,5 Mb	4,6 Mb	3,6 Mb
	Pourcentage de bases erronées	62,23 %	53,18 %	3,63 %	0,75 %
	Pourcentage de bases manquantes	50,22 %	17,16 %	0,60 %	0,11 %

Le Tableau 3.5 détaille les résultats obtenus avec ABySS. Comme Velvet et ABySS sont tous les deux des assembleurs de de Bruijn, la qualité des séquences assemblées avec ABySS est très similaire à celle obtenue avec Velvet. Nous obtenons également des contigs et des scaffolds de meilleure qualité en augmentant la longueur des k-mers pour les mêmes raisons que celles mentionnées pour Velvet. Néanmoins, nous pouvons observer une légère amélioration de la couverture des séquences de référence par les contigs ou les scaffolds, ainsi qu'une faible augmentation du pourcentage d'erreurs d'assemblage. Cela est probablement dû aux choix effectués par ABySS dans le processus d'assemblage qui préfère étendre davantage les contigs en utilisant des contraintes plus faibles.

La qualité des résultats diffère significativement entre les trois approches d'assemblage. La qualité de l'assemblage obtenu avec l'assembleur Greedy SSAKE est significativement plus faible en comparaison avec celle des assembleurs basés sur l'approche Overlap-Layout-Consensus ou de de Bruijn, comme présenté dans [104]. Comme indiqué dans la Section 2.1.2, les meilleurs résultats pour les deux organismes sont obtenus

TABLEAU 3.5 – Qualité globale de l'assemblage avec ABySS

Longueur k-mers		**17 bp**	**27 bp**	**37 bp**	**47 bp**
Contigs C. elegans chromosome I	**Nombre de séquences**	936.997	49.252	24.674	19.534
	N50	64	3.038	4.968	2.990
	Longueur totale	27,7 Mb	15,4 Mb	15,2 Mb	15,2 Mb
	Pourcentage de bases erronées	73,00 %	8,38 %	3,33 %	1,91 %
	Pourcentage de bases manquantes	51,16 %	5,40 %	0,75 %	0,35 %
Scaffolds C. elegans chromosome I	**Nombre de séquences**	936.903	48.974	24.525	19.484
	N50	64	3.162	5.049	3.012
	Longueur totale	27,7 Mb	15,4 Mb	15,2 Mb	15,2 Mb
	Pourcentage de bases erronées	73,01 %	8,42 %	3,36 %	1,92 %
	Pourcentage de bases manquantes	51,18 %	5,39 %	0,75 %	0,35 %
Contigs L. pneumophila souche Philadelphia	**Nombre de séquences**	12.140	441	135	128
	N50	1.756	220.550	220.559	230.276
	Longueur totale	3,6 Mb	3,5 Mb	3,4 Mb	3,4 Mb
	Pourcentage de bases erronées	5,03 %	0,52 %	0,23 %	0,21 %
	Pourcentage de bases manquantes	1,49 %	0,12 %	0,01 %	0,01 %
Scaffolds L. pneumophila souche Philadelphia	**Nombre de séquences**	10.712	435	132	98
	N50	6.079	229.920	250.977	250.612
	Longueur totale	3,7 Mb	3,5 Mb	3,4 Mb	3,4 Mb
	Pourcentage de bases erronées	7,55 %	0,53 %	0,24 %	0,18 %
	Pourcentage de bases manquantes	1,65 %	0,14 %	0,24 %	0 %

avec les assembleurs basés sur l'approche de de Bruijn, ABySS et Velvet.

3.3.2 Qualité de la résolution des répétitions en tandem

En analysant avec GAGE la qualité globale des séquences assemblées obtenues par ABySS et Velvet, nous pouvons déduire que l'ensemble des bases du premier chromosome de *C. elegans* et du génome de *L. pneumophila* ont été couvertes. Cela est également valable pour SGA sur le génome de *L. pneumophila*. Cependant, GAGE calcule le pourcentage de bases erronées ou manquantes en alignant des sous-séquences des contigs et scaffolds sur les séquences de références. Ce type de calcul implique que deux sous-séquences adjacentes des contigs/scaffolds peuvent ne pas s'aligner de façon adjacente sur la séquence de référence, mais les bases sont néanmoins considérées comme correctes. En conséquence, avec ce type d'analyse nous ne pouvons pas déduire la qualité de la résolution des RT. De ce fait, nous présentons également une analyse axée uniquement sur les RT.

Nous utilisons mreps [60] pour identifier les RT de chaque séquence de référence ainsi que celles des contigs et scaffolds obtenus par les assembleurs.

Dans notre analyse, nous considérons pour chaque fragment d'ADN cible D seulement son brin de référence et nous identifions les RT que les assembleurs doivent résoudre en exécutant mreps. Ensuite, chaque RT détectée dans les contigs ou les scaffolds obtenus par les assembleurs peut être classée parmi les catégories suivantes :

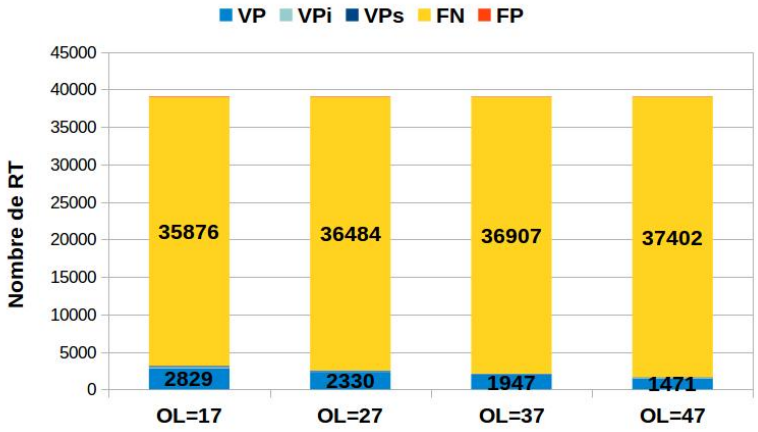
- Vrai Positif (VP) si nous identifions dans les contigs/scaffolds la séquence complète (c'est-à-dire avec le nombre correct de copies) de la RT de D ;
- Vrai Positif incomplet (VP_i) si nous identifions dans les contigs/scaffolds seulement une partie de la RT de D ;
- Vrai Positif sur-étendu (VP_s) si nous identifions dans les contigs/scaffolds la RT de D avec un nombre supérieur de copies ;
- Faux Négatif (FN) si nous n'identifions pas dans les contigs/scaffolds la RT de D ;
- Faux Positif (FP) si nous identifions dans les contigs/scaffolds une RT, mais sa séquence n'apparaît pas dans D.

La qualité des résultats obtenus par chaque assembleur est ensuite mesurée en utilisant les deux statistiques suivantes :

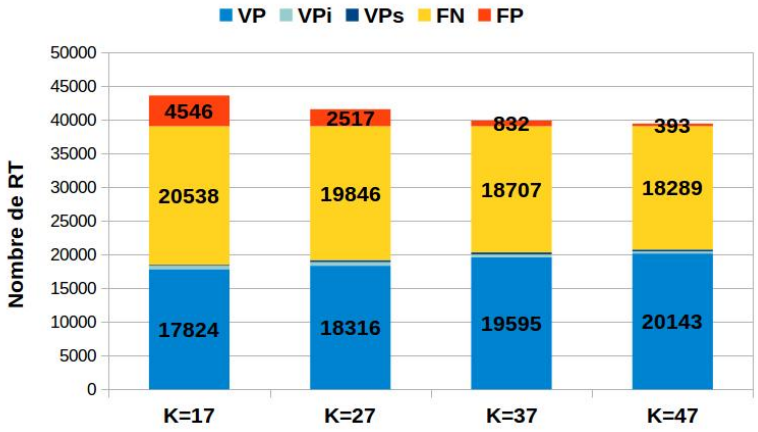
- *Précision* $= (VP + VP_i + VP_s)/(VP + VP_i + VP_s + FP)$ qui mesure la fraction des RT récupérées qui sont présentes dans D ;
- *Sensibilité* $= (VP + VP_i + VP_s)/(VP + VP_i + VP_s + FN)$ qui mesure la fraction des RT de D qui sont correctement identifiées.

Pour notre analyse, nous considérons les RT avec une longueur de motifs de maximum 100 bp et nous avons identifié avec mreps 39006 RT sur la séquence du premier chromosome de *C. elegans* et 2227 RT sur la séquence du génome de *L. pneumophila* (souche *Philadelphia*). La qualité des résultats obtenus avec SSAKE est très faible. Pour chaque organisme testé, l'assembleur ne parvient pas à assembler plus de 1% de RT. En conséquence, nous détaillons par la suite les résultats obtenus avec les trois autres assembleurs, SGA, Velvet et ABySS.

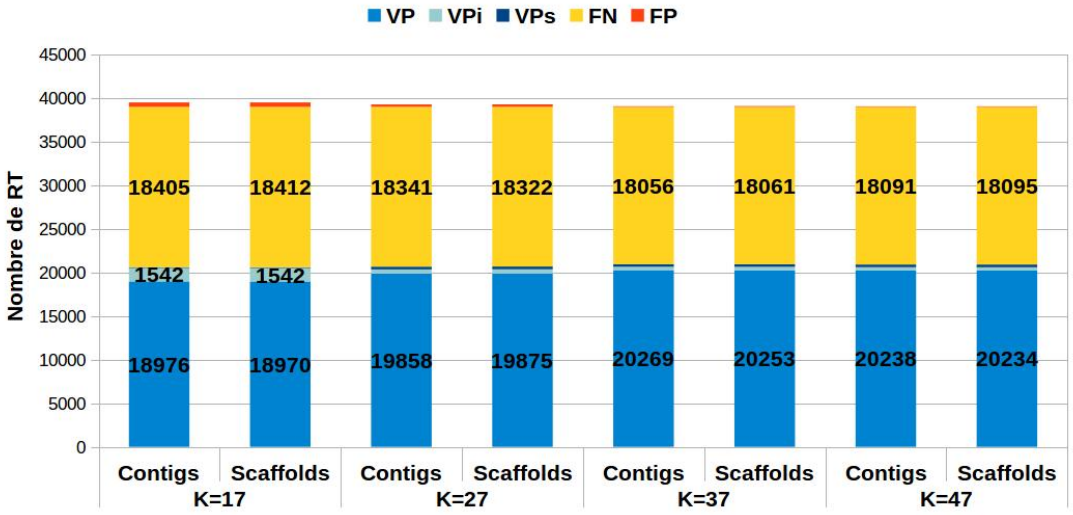
La Figure 3.3 présente les résultats obtenus par les trois assembleurs quant au nombre de RT correctement assemblées du premier chromosome de *C. elegans*. Nous pouvons observer qu'un nombre important de RT n'est pas présent dans les contigs ou scaffolds des assembleurs. Cela est particulièrement vrai pour les séquences assemblées par SGA (voir Figure 3.3a). Comme présenté dans la précédente section, nous avons varié pour SGA la longueur minimum d'un chevauchement (*OL*) entre 17 et 47. Néanmoins, l'assembleur ne réussit pas à assembler plus de 7,3% de RT du premier chromosome de *C. elegans*. Comme présenté dans l'analyse globale de la qualité de ses scaffolds, SGA ne réussit à assembler que très peu de séquences. Durant le processus d'assemblage, SGA utilise des contraintes fortes dans les étapes Layout et Consensus, comme, par exemple, le nombre de reads validant une position dans la séquence de consensus. Ainsi, de nombreux chemins ne peuvent pas être linéarisés en respectant les contraintes de validation de chemins et SGA décide de laisser ces parties du graphe de chevauchement non-assemblées.



(a) Nombre de RT détectées par SGA



(b) Nombre de RT détectées par Velvet



(c) Nombre de RT détectées par ABySS

FIGURE 3.3 – Nombre de RT détectées par les assembleurs pour le premier chromosome de *C. elegans*

Les valeurs de la sensibilité sont donc très basses, mais comme l'assembleur obtient un nombre très faible de FP, les valeurs de la précision sont supérieures à 0,966 (voir deuxième et troisième colonnes du Tableau 3.6). Ce faible taux de FP est également dû aux critères utilisés pendant les étapes Layout et Consensus. En étant très strict avec le nombre de reads validant une séquence assemblée, SGA fait donc très peu d'erreurs.

TABLEAU 3.6 – Précision et sensibilité de la détection des RT avec SGA pour le premier chromosome de *C. elegans* et pour le génome de *L. pneumophila* (souche *Philadelphia*).

	Ier Chromosome de C. elegans		**Génome de L. pneumophila**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
K=17	**0,976**	**0,080**	0,988	0,357
K=27	0,975	0,065	0,991	0,488
K=37	0,974	0,054	**0,993**	0,599
K=47	0,966	0,041	**0,993**	**0,614**

Dans les Figures 3.3b et 3.3c nous pouvons observer que les assembleurs de de Bruijn obtiennent des meilleurs résultats que l'assembleur basé sur l'approche OLC. La fragmentation des reads dans des k-mers permet aux assembleurs de de Bruijn de mieux résoudre les RT courtes. En matière de vrais positifs, le nombre de RT varie de moins de 10% entre les résultats obtenus avec k=17 et k=47. Cette observation est valable pour les deux assembleurs de de Bruijn. Cependant, la proportion de RTE dans les VP est très différente en fonction de la longueur des k-mers. Entre k=17 et k=47, le nombre de RTE correctement identifiées décroit de plus de 50% pour les deux assembleurs de Bruijn.

La différence de la qualité des résultats entre les deux assembleurs de de Bruijn est très faible. Néanmoins, les valeurs de la précision et de la sensibilité obtenues pour les contigs ou les scaffolds d'ABySS (voir Tableau 3.8) sont plus élevées que celles obtenues pour les scaffolds de Velvet (voir deuxième et troisième colonnes du Tableau 3.7).

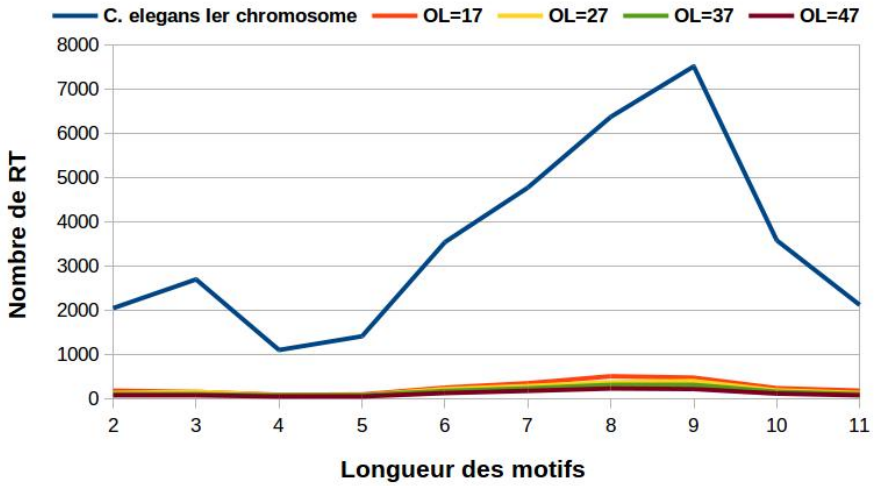
TABLEAU 3.7 – Précision et sensibilité de la détection des RT avec Velvet pour le premier chromosome de *C. elegans* et pour le génome de *L. pneumophila* (souche *Philadelphia*).

	Ier Chromosome de C. elegans		**Génome de L. pneumophila**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
K=17	0,802	0,473	0,219	0,476
K=27	0,884	0,491	0,337	0,498
K=37	0,961	0,520	0,852	0,539
K=47	**0,981**	**0,531**	**0,960**	**0,545**

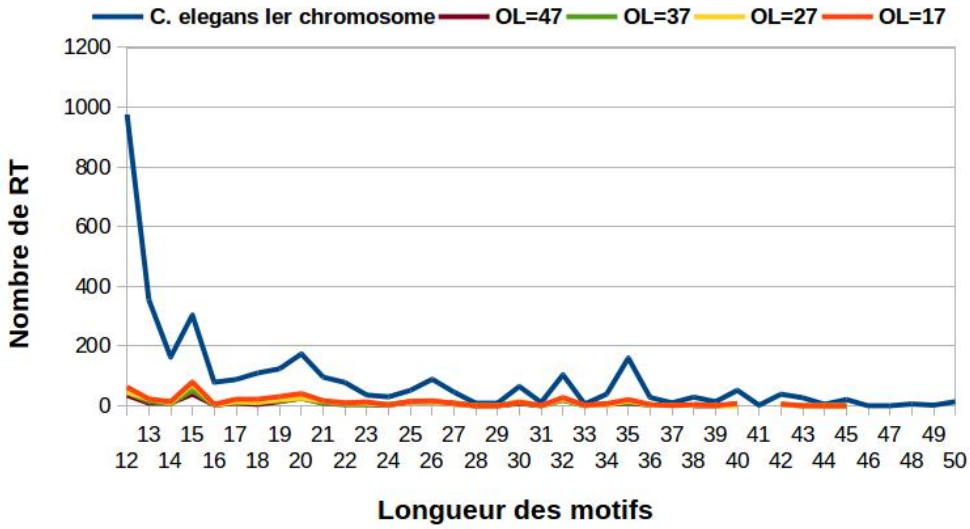
Les Figures 3.4, 3.5 et 3.6 présentent la distribution de la longueur des motifs des RT du premier chromosome de *C. elegans* ainsi que des RT assemblées dans les contigs ou les scaffolds avec respectivement SGA, Velvet et ABySS pour ce chromosome. Nous pouvons observer que les assembleurs ne réussissent pas à résoudre des RT avec des motifs longs. Dans les ensembles de séquences assemblées, mreps ne détecte pas de RT avec des motifs plus longs que 45 bp, tandis que, dans la séquence de référence du premier chromosome de *C. elegans*, il détecte des RT avec des motifs allant jusqu'à 100 bp.

TABLEAU 3.8 – Précision et sensibilité de la détection des RT avec ABySS pour le premier chromosome de *C. elegans*.

	Contigs		Scaffolds	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
K=17	0,976	0,528	0,976	0,528
K=27	0,987	0,530	0,987	0,530
K=37	**0,997**	**0,537**	0,996	**0,537**
K=47	0,987	0,536	**0,997**	0,536



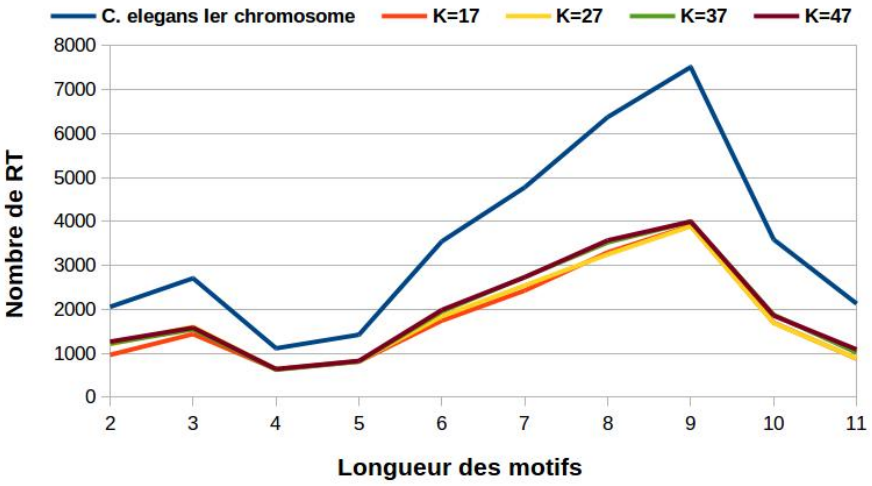
(a) Résultats obtenus pour les microsatellites



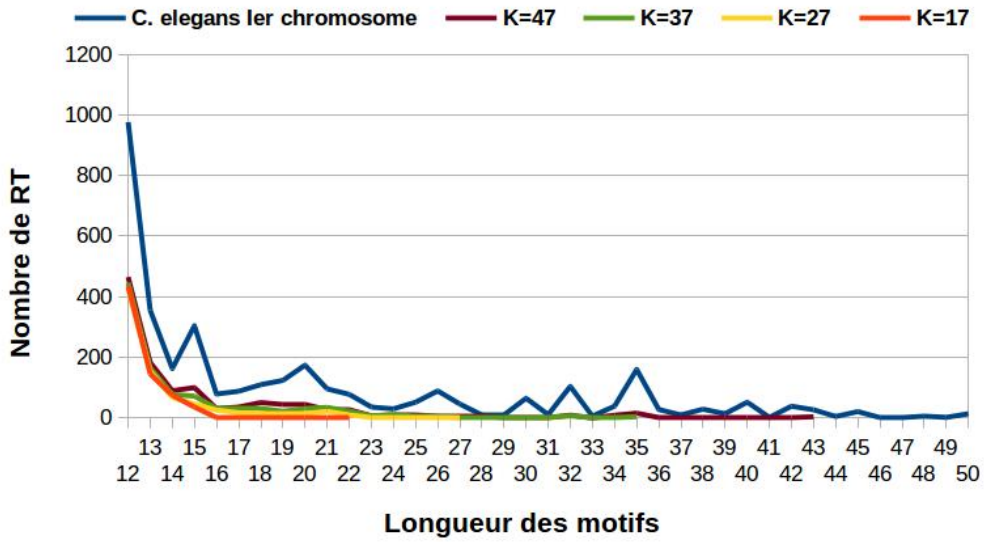
(b) Résultats obtenus pour les minisatellites

FIGURE 3.4 – Distribution de la longueur des motifs des RT détectées avec SGA pour le premier chromosome de *C. elegans*

De façon générale, nous pouvons observer que les assembleurs de de Bruijn ne résolvent pas de RT avec des motifs plus longs que k bp. La seule exception a été observée pour Velvet dans le cas $k = 17$ où la longueur maximum des motifs de RT est 22 bp.



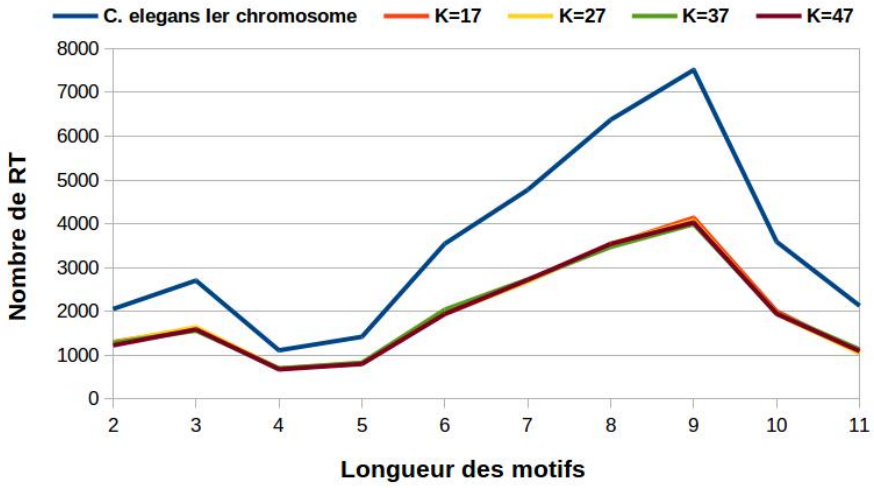
(a) Résultats obtenus pour les microsatellites



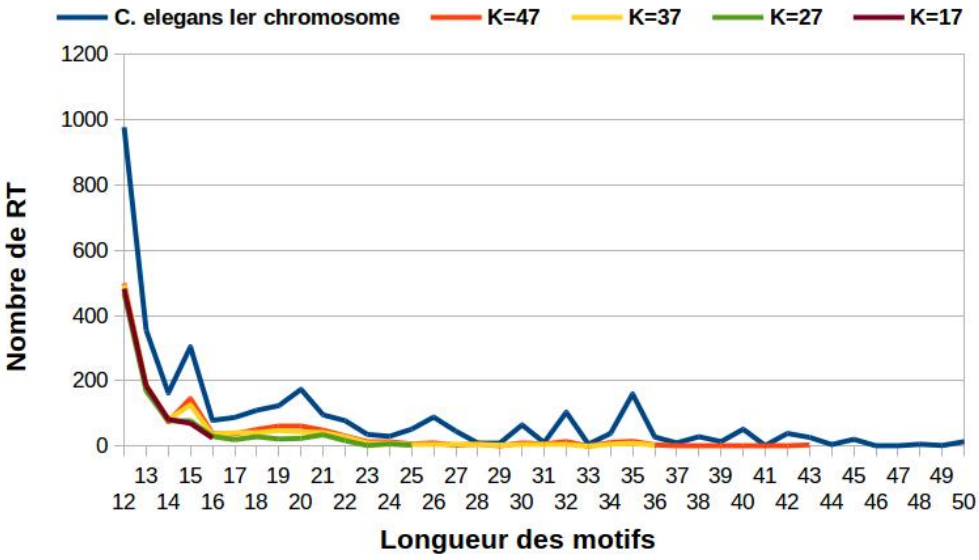
(b) Résultats obtenus pour les minisatellites

FIGURE 3.5 – Distribution de la longueur des motifs des RT détectées avec Velvet pour le premier chromosome de *C. elegans*

La Figure 3.7 présente les résultats obtenus par les trois assembleurs quant au nombre de RT correctement assemblées du génome de *L. pneumophila* (souche *Philadelphia*). Dans le cas de SGA et ABySS les résultats sont meilleurs que pour le premier chromosome de *C. elegans*. Les deux assembleurs arrivent à résoudre approximativement 60% des RT du génome. De plus, le nombre de faux positifs est très faible comme le montrent les valeurs de la précision (voir Tableaux 3.6 et 3.9). Inversement, Velvet résout moins de RT que les deux autres assembleurs et fait de nombreuses erreurs d'assemblage. Pour k=17, le nombre de faux positifs est 3,6 fois plus élevé que le nombre de vrais positifs.



(a) Résultats obtenus pour les microsatellites

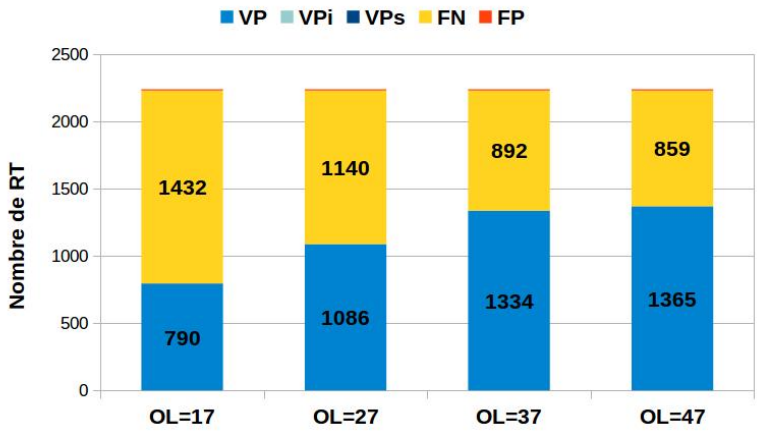


(b) Résultats obtenus pour les minisatellites

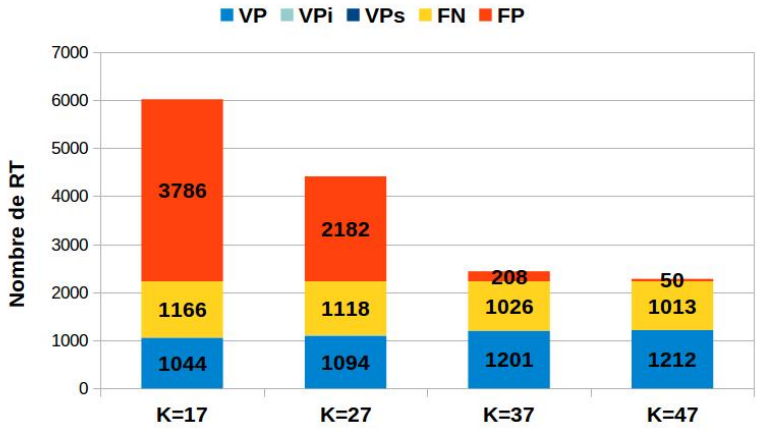
FIGURE 3.6 – Distribution de la longueur des motifs des RT détectées avec ABySS pour le premier chromosome de *C. elegans*

TABLEAU 3.9 – Précision et sensibilité de la détection des RT avec ABySS pour le génome de *L. pneumophila* (souche *Philadelphia*).

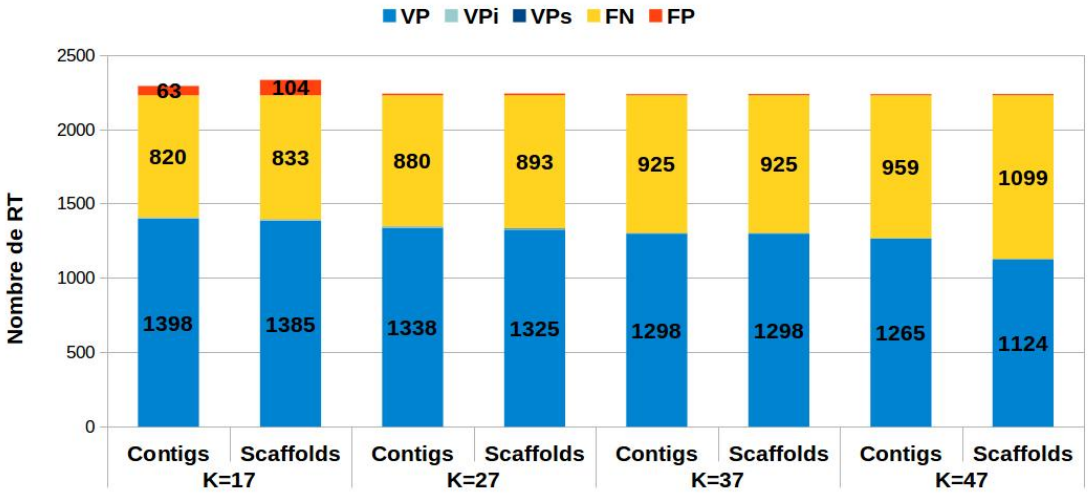
	Contigs		**Scaffolds**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
K=17	0,957	**0,632**	0,931	**0,626**
K=27	0,992	0,605	0,991	0,599
K=37	**0,994**	0,585	**0,993**	0,585
K=47	**0,994**	0,569	0,992	0,507



(a) Nombre de RT détectées par SGA



(b) Nombre de RT détectées par Velvet

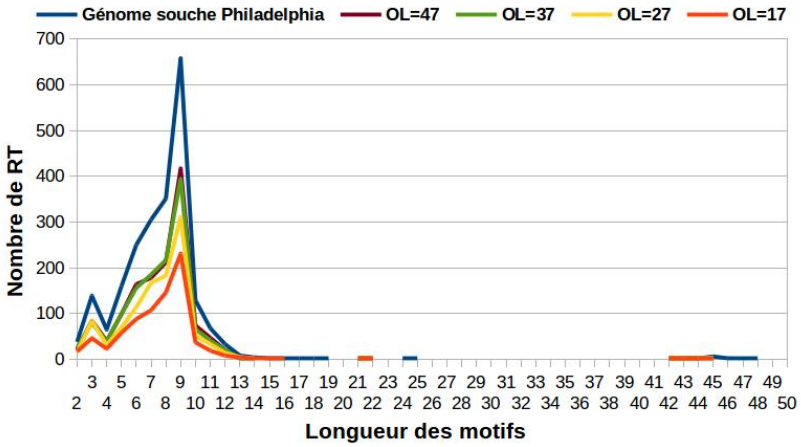


(c) Nombre de RT détectées par ABySS

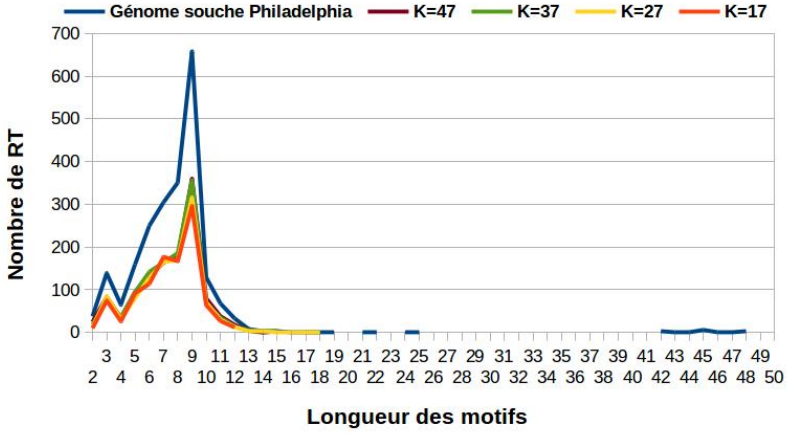
FIGURE 3.7 – Nombre de RT détectées par les assembleurs pour le génome de *L. pneumophila* (souche *Philadelphia*)

La distribution de la longueur des motifs pour les RT du génome de la souche *Philadelphia* du *L. pneumophila* et celle des RT assemblées dans les contigs/scaffolds par les trois assembleurs sont présentées dans la Figure 3.8. Comme dans le cas du premier chromosome de *C. elegans*, les assembleurs n'arrivent pas à assembler des RT avec des

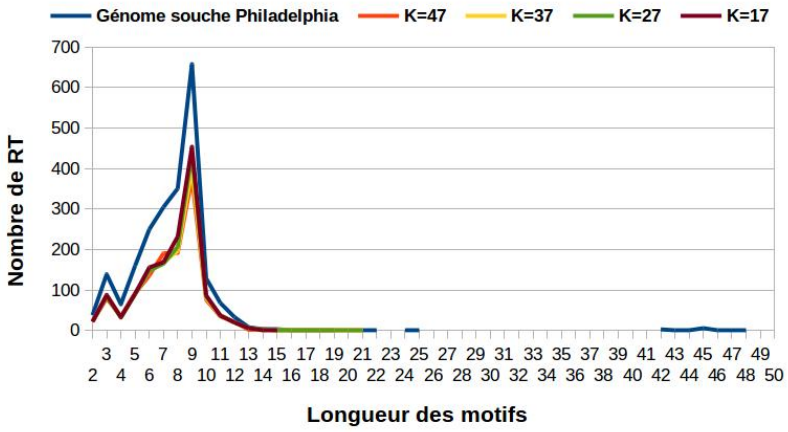
motifs longs. Les assembleurs de de Bruijn ne résolvent pas de RT avec des motifs plus longs que k bp, et, même avec des valeurs de $k \geq 25$, ils ne peuvent pas assembler des RT avec des motifs plus longs que 21 bp. Seul l'assembleur SGA réussit à résoudre des RT avec des motifs jusqu'à 45 bp.



(a) Résultats obtenus avec SGA



(b) Résultats obtenus avec Velvet



(c) Résultats obtenus avec ABySS

FIGURE 3.8 – Distribution de la longueur des motifs des RT détectées par les assembleurs pour le génome de *L. pneumophila* (souche *Philadelphia*)

Plusieurs RT de la souche *Philadelphia* de *L. pneumophila* ont été étudiées pour leur signification biologique [23, 121]. Nous présentons dans le Tableau 3.10 les résultats obtenus par les quatre assembleurs pour toutes les RT présentées dans [23, 121] et ayant une longueur de motif de $2 \leq |p| \leq 100$.

TABLEAU 3.10 – Résultats des assembleurs concernant la résolution des RT biologiquement significatives pour le génome de *L. pneumophila* (la souche *Philadelphia*) présentées dans [23, 121].

Nom	**Gène**	**Longueur du motif**	**Nombre de copies**	**SSAKE**	**SGA**	**ABySS**	**Velvet**
	LPG0451	30 bp	5.9	-	-	-	-
	LPG0688	9 bp	6	-	-	Complète (K=17,37, 47)	-
	LPG1038	12 bp	4.17	-	Complète (OL=37,47)	Complète (K=17)	-
Lpms35	LPG1299	18 bp	3	-	-	-	-
	LPG1555	21 bp	2	-	Complète (OL=17,27, 37,47)	Complète (K=27,37, 47)	-
	LPG1602	90 bp	9.2	-	Complète (OL=17,27, 37,47)	Complète (K=17,27, 37,47)	-
	LPG1948	90 bp	7.08	-	Incomplète (5 copies OL=27)	-	-
	LPG1958	87 bp	13.59	-	-	Incomplète (6 copies K=27,37,47)	-
	LPG2392	87 bp	6.49	-	-	-	-
	LPG2559	12 bp	4.08	-	Complète (OL=17,37, 47)	-	-
Lpms31	LPG2644	45 bp	19.44	-	-	-	-
Lpms3	LPG2793	96 bp	7.58	-	-	Complète (K=27,37)	-
Lpms01	LPG2854	45 bp	7.64	-	-	-	-
Lpms13	LPG1488	24 bp	9.75	-	-	-	-
Lpms17	LPG0854	89 bp	2.28	-	-	-	-
Lpms19	Intergenic	21 bp	4.05	-	Incomplète (2 copies OL=37)	-	-

Nous pouvons observer que ni SSAKE, ni Velvet ne parviennent à assembler ces RT. C'est seulement dans les contigs/scaffolds de SGA et ABySS que nous pouvons identifier quelques RT significatives. Cependant, plus de la moitié des RT ne sont pas détectées par

les deux assembleurs. Celles qui sont correctement assemblées ont été détectées pour des valeurs différentes de k ou de la longueur minimum du chevauchement (OL).

Les résultats obtenus pendant nos expériences soulignent l'insuffisance des solutions existantes pour le problème de l'assemblage des RT. Comme présenté dans la littérature [117, 132], les assembleurs de novo s'efforcent d'améliorer la qualité de la résolution des répétitions, mais ils ne réussissent pas à les assembler correctement dans la plupart des cas. De plus, l'objectif principal de ces outils est l'assemblage global des fragments d'ADN cibles. Ainsi les heuristiques utilisées font souvent des choix en défaveur des RT. C'est pour ces raisons que nous avons concentré notre travail sur l'amélioration de l'assemblage de novo de RT en proposant des outils adaptés à ce problème.

3.4 Représentation des répétitions en tandem exactes dans le graphe de de Bruijn

Dans cette section, nous présentons les particularités de la représentation des RTE dans un graphe de de Bruijn qui est la structure de données sur laquelle sont basés les algorithmes proposés dans nos travaux.

Soit k un entier positif et ε une RTE du motif p telle que $|\varepsilon| \geq |p| + k$ et que la séquence $Pref(\varepsilon, |p| + k - 1)$ ne contient aucune répétition exacte, dispersée ou en tandem, d'un motif q pour $|q| \geq k$. Soit $S^k(\varepsilon) = \{v_1, v_2, \ldots, v_{|p|}\}$ le k-spectre de ε tel que $v_i = \varepsilon[i, k]$, $1 \leq i \leq |p|$. L'ensemble $S^k(\varepsilon)$ est composé seulement des $|p|$ premiers k-mers de ε. En effet, comme à chaque $|p|$-ième position dans ε on retrouve p ou un préfixe de p, après $|p|$ bp en ε on obtient des k-mers qui sont déjà présents dans $S^k(\varepsilon)$. De plus, comme la séquence $Pref(\varepsilon, |p| + k - 1)$ ne contient aucune répétition exacte, dispersée ou en tandem, d'un motif q pour $|q| \geq k$, chacun de ses k-mers est unique et donc $|S^k(\varepsilon)| = |p|$. Le graphe de de Bruijn $G^k(\varepsilon)$, construit à partir de $S^k(\varepsilon)$, consiste donc en un cycle c élémentaire (c'est-à-dire, pour former c, chaque nœud, excepté le nœud extrémité, est traversé une seule fois) où ses nœuds sont reliés comme démontré par la suite. Un chemin élémentaire est construit à partir de v_1 jusqu'à $v_{|p|}$ comme $Suff(v_i, k-1) = Pref(v_{i+1}, k-1)$, pour $1 \leq i < |p|$. À cause de la condition $|\varepsilon| \geq |p| + k$, on déduit que $\varepsilon[p+1, k] = p[1, k] = \varepsilon[1, k]$ et donc que $Suff(v_{|p|}, k-1) = Pref(v_1, k-1)$. Par conséquent, $v_{|p|}$ est aussi connecté à v_1 par l'arc $(v_{|p|}, v_1)$. Un exemple est présenté sur la Figure 3.9.

Néanmoins, si les deux conditions sur la longueur minimale de la RTE et sur l'absence de répétitions internes ayant des motifs de longueur minimale k bp ne sont pas satisfaites, la RTE ne forme pas un cycle élémentaire. Un exemple de RTE contenant une répétition interne avec un motif de longueur k est présenté dans la Figure 3.10. Par la suite nous allons nous intéresser aux RTE respectant ces deux conditions.

Comme dans le graphe de de Bruijn chaque k-mer est représenté une seule fois, on a besoin de connaître le nombre de fois qu'un arc va être traversé pour effectuer un assemblage. Pour cela, on va utiliser la notion de *fréquence d'arc*.

Nous rappelons qu'une séquence s_1 de longueur l est *présente* dans une séquence s_2 s'il existe une position $1 \leq i \leq (|s_2| - l + 1)$ telle que $s_1 = s_2[i, l]$. Le *nombre d'occurrences* $occ(s_1, s_2)$ d'une séquence s_1 dans une séquence s_2 est représenté par le nombre de positions i pour lesquelles $s_1 = s_2[i, l]$. Dans le cas d'un ensemble de séquences, le *nombre d'occurrences* $occ(s, \zeta)$ d'une séquence s dans un ensemble ζ de séquences est égal au $\sum_{t \in \zeta} occ(s, t)$.

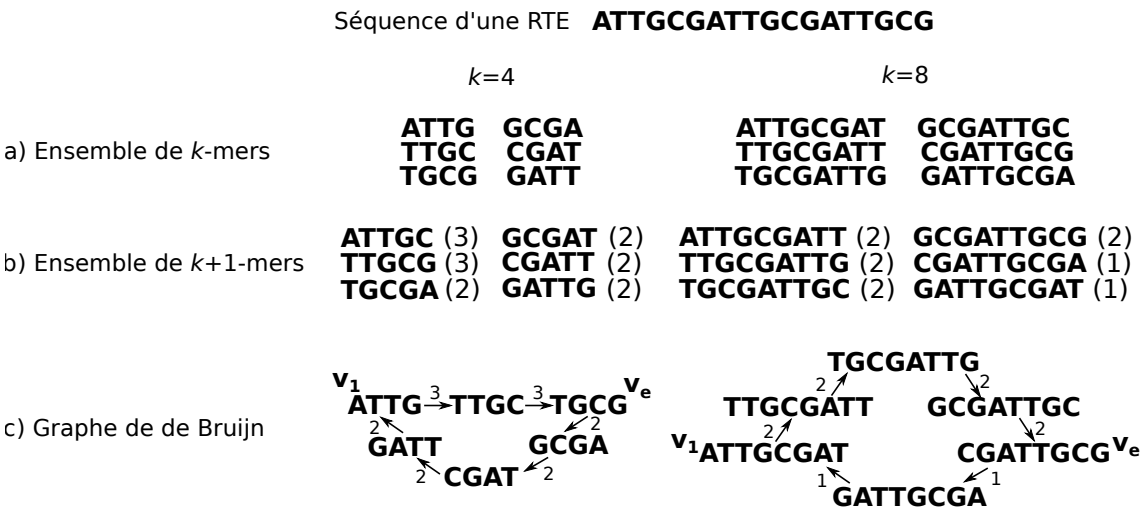


FIGURE 3.9 – Le graphe de de Bruijn d'une RTE avec p=**ATTGCG** et cn_p=3 pour deux cas : $|p|>k$ (k=4, à gauche) et $|p|\leq k$ (k=8, à droite). Les nœuds v_1 et v_e sont ceux décrits dans la Propriété 1. La fréquence pour chaque arc du graphe de de Bruijn est donnée par le nombre d'occurrences du (k+1)-mer correspondant dans la RTE (donnée entre parenthèses au b)). Par souci de simplification, nous choisissons pour la couverture $\delta = 1$.

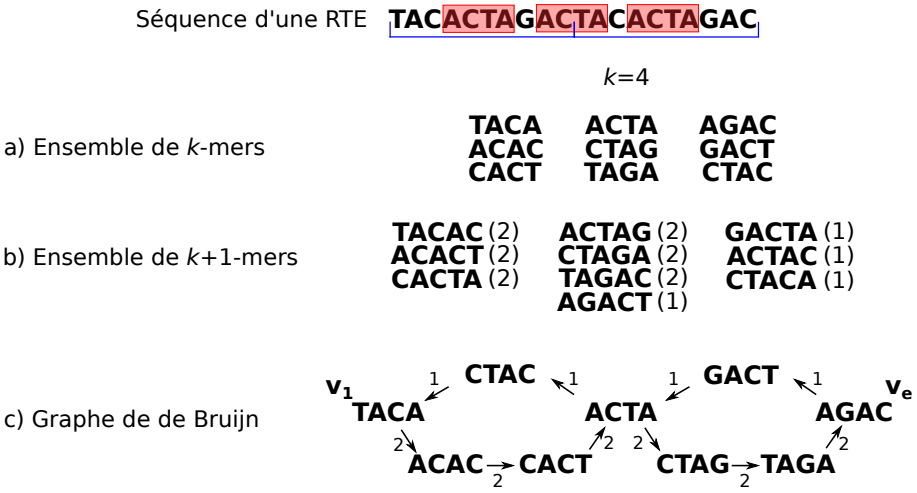


FIGURE 3.10 – Le graphe de de Bruijn d'une RTE avec p=**TACACTAGAC** et cn_p=2 (soulignée en bleu) qui contient une répétition interne dispersée du motif q=**ACTA** (soulignée en rouge). Comme $k = |q|$, le k-mer **ACTA** apparaît 2 fois dans le préfixe de la RTE de longueur $|p| + k - 1$ et le cycle formé par la RTE n'est pas élémentaire.

Définition 13 (Fréquence d'un arc)**.** *Soient le fragment d'ADN cible D, l'ensemble de reads courts R obtenu après le séquençage de D et le graphe de de Bruijn $G^k(R)$. La* fréquence d'un arc $\alpha = (v_i, v_j)$ dans $G^k(R)$ *est égale à* $f(\alpha, R) = occ(v, R)/\delta$ *où* $v = v_i + v_j[k]$ *et δ représente la couverture de R.*

En effet, on considère que la fréquence d'un arc est égale au nombre de fois où l'arc doit être traversé pour assembler D. En outre, pour chaque cycle correspondant à une RTE, la Propriété 1 ci-dessous est satisfaite (voir aussi la Figure 3.9).

Propriété 1 (Propriété des fréquences des arcs pour le cycle d'une RTE)**.** *Soient* $v_1 = \varepsilon[1,k]$ *et* $v_e = Suff(\varepsilon,k)$*, respectivement les premiers et derniers* k*-mers d'une RTE* $\varepsilon = (p, cn_p)$ *telle que* $|\varepsilon| \geq |p| + k$ *et que la séquence* $Pref(\varepsilon, |p| + k - 1)$ *ne contient pas de répétition exacte, dispersée ou en tandem, d'un motif* q *pour* $|q| \geq k$*. Dans le cycle élémentaire formé par* ε *dans* $G^k(\varepsilon)$*, la fréquence de chaque arc du chemin de* v_1 *à* v_e *est la même. Si* x *désigne cette fréquence, alors la fréquence de chaque arc du chemin de* v_e *à* v_1 *est égale à* $x - 1$*.*

La Propriété 1 est respectée dans les deux cas, pour $|p| \leq k$ et $|p| > k$ comme présenté sur la Figure 3.9 pour deux valeurs différentes de k. Dans les deux cas, le nombre de nœuds dans le cycle est égal à $|p|$ en raison de la présence unique de chaque k-mer dans le graphe de de Bruijn.

Néanmoins, dans un graphe de de Bruijn $G^k(R)$, un cycle n'est pas nécessairement formé par une RTE. Toute répétition exacte appartenant au fragment d'ADN cible avec un motif d'une longueur d'au moins k bp forme un cycle. Cela est valable pour les RTE mais également pour les répétitions dispersées exactes. Deux k-mers du fragment d'ADN cible sont reliés par un chemin dans le graphe de de Bruijn. Mais, comme dans ce graphe, chaque nœud représente un k-mer unique, si ces deux k-mers sont identiques alors ils sont représentés par un même nœud et le chemin les reliant est en fait un cycle.

De plus, un cycle formé par une répétition dispersée exacte peut avoir les mêmes types de fréquences que celles énoncées par la Propriété 1. C'est le cas d'une répétition dispersée avec deux copies exactes d'un motif. Soit un fragment d'ADN s contenant une répétition dispersée exacte du motif primitif t d'une longueur $|t| \geq k$ telle que $s = t+\beta+t$ où $|\beta| \geq 1$ et $\beta \neq Pref(t,i)$ pour $1 \leq i \leq |p|$. Nous supposons que la séquence β est primitive et qu'aucun k-mer de β n'est présent dans t. Ainsi, le nombre d'occurrences pour chaque k-mer κ_i de la séquence β vérifie les conditions $occ(\kappa_i, \beta) = 1$ et $occ(\kappa_i, t) = 0$. Nous considérons $v_1 = s[1,k]$ et $v_e = Suff(s,k)$, comme dans la Propriété 1. Comme $occ(\kappa_i, \beta) = 1$ et $occ(\kappa_i, t) = 0$, alors la fréquence de chaque arc du chemin de v_e à v_1, représentant la séquence β, est 1. Comme le motif t est primitif, alors la fréquence de chaque arc du chemin de v_1 à v_e, représentant le motif t, est égale au nombre de copies de t dans s, c'est-à-dire 2. En conséquence, si le motif t et la séquence β sont primitifs et n'ont aucun k-mer en commun, alors la Propriété 1 est vérifiée pour le cycle formé par la séquence s.

Cependant, si le nombre de copies de t est supérieur à 2, la Propriété 1 n'est pas vérifiée. Dans ce cas $s = t + \beta_1 + t + \ldots + t + \beta_n + t$ où $|\beta_j| \geq 1$ et $\beta_j \neq Pref(t,i)$ pour $1 \leq i \leq |p|$ et $1 \leq j \leq n$. Comme s n'est pas une RTE alors $\beta_i \neq \beta_j$ pour $i \neq j$. En conséquence, chaque sous-séquence $t+\beta_i+t$ de s forme un cycle différent. La fréquence de chaque arc du chemin de v_1 à v_e, représentant le motif t, est égale au nombre de copies de t dans s, c'est-à-dire $n+1$, et chaque β_i est représenté par un chemin différent entre v_e et v_1 où la fréquence d'au moins un arc est inférieure à n.

3.5 Conclusion

En raison du rôle biologique important des RT, le problème de la détection des RT a fait l'objet de nombreuses études. Si la détection des RT dans des séquences d'ADN assemblées peut être effectuée par un nombre important d'outils existants, l'assemblage des RT dans un contexte de novo reste un problème difficile [117]. Les assembleurs de novo représentent les principaux outils qui permettent la résolution des RT à partir des

reads et sans utiliser une séquence de référence. Cependant, dans leur quête vers des séquences assemblées les plus longues possible et de haute qualité, ils peuvent laisser de nombreuses répétitions, en particulier des RT, non résolues en raison de la complexité des génomes [117].

Parmi les approches des assembleurs de novo, les meilleurs résultats pour l'assemblage des RT sont obtenus par ceux utilisant une approche de de Bruijn. De plus, le graphe de de Bruijn permet la représentation de certaines RTE avec un modèle topologique très spécifique et donc facilement identifiable. C'est pourquoi les algorithmes que nous proposons par la suite pour améliorer la résolution des RT sont basés sur ce type de graphe. Notre première contribution, l'algorithme DExTaR, a pour but d'assembler des RTE non-résolues suite à un assemblage de de Bruijn global. Le deuxième algorithme, appelé MixTaR, est dédié à l'assemblage des RT en général en partant directement du graphe de de Bruijn non-assemblé. Dans les deux chapitres qui suivent, nous présentons ces algorithmes et les résultats obtenus.

4

Algorithme de détection des répétitions en tandem exactes

Dans les chapitres précédents nous avons souligné le fait que l'assemblage de RT est une étape très difficile. Cela est causé par la complexité des RT (les différents types de RT, le nombre variable de copies d'un motif dans une RT, les répétitions possibles d'une RT ou d'une partie de celle-ci dans différentes parties du génome), mais également par les limites des technologies actuelles de séquençage (par exemple, la non-homogénéité de la couverture). Par conséquent, les assembleurs ne sont pas toujours en mesure de trouver une séquence de consensus pour les parties contenant les RT, et ne peuvent donc pas les assembler. Notre travail, présenté dans ce chapitre, vise à améliorer l'assemblage des RT en se focalisant sur un type de RT spécifique, les RTE. L'algorithme, appelé DExTaR, a pour but de compléter et corriger l'ensemble des RTE détectées suite à un assemblage de de Bruijn, en analysant les parties qui restent non résolues.

Dans ce chapitre, nous commençons par présenter en détail l'algorithme DExTaR dans le cas théorique de données parfaites : des reads courts sans erreurs et avec une couverture homogène. Ensuite, nous exposons l'adaptation de l'algorithme pour le cas réel et l'analyse des résultats obtenus. Ces travaux ont été réalisés en collaboration avec Guillaume Fertin, Géraldine Jean et Irena Rusu, une partie étant publiée dans [35].

4.1 Description détaillée de DExTaR

Notre algorithme DExTaR a été conçu pour trouver les RTE laissées non-résolues par un assemblage basé sur l'approche de de Bruijn. Nous rappelons qu'un assembleur de de Bruijn modifie à chaque étape le graphe de de Bruijn dans le but d'avoir des séquences plus longues dans les nœuds, tout d'abord des k-mers, ensuite des unitigs et enfin des contigs. DExTaR recherche les RTE non-résolues dans le graphe construit par un assembleur à la fin d'une étape d'assemblage. L'algorithme peut être appliqué sur le graphe résultant soit à la fin de l'assemblage de unitigs, appelé *graphe d'unitigs*, soit à la fin de l'assemblage des contigs, appelé *graphe de contigs*.

DExTaR représente une solution au Problème de DÉTECTION DE NOVO DE RTE SUITE À UN ASSEMBLAGE DE DE BRUIJN qui est défini comme suit :

DÉTECTION DE NOVO DE RTE SUITE À UN ASSEMBLAGE DE DE BRUIJN

Entrée : Un ensemble *R* de reads courts obtenu suite à un processus de séquençage d'un fragment d'ADN cible D et le graphe $G^k_{u/c}(R)$ d'unitigs ou contigs obtenu à partir de *R* par un assembleur de de Bruijn.

Objectif : Trouver l'ensemble des RTE du fragment D qui n'ont pas été assemblées dans le graphe $G^k_{u/c}(R)$.

Notre algorithme parcourt le graphe d'unitigs ou de contigs pour retrouver des cycles qui, une fois assemblés, permettent l'identification des RTE non-résolues par l'assembleur. Comme précisé auparavant, le cycle formé par une RTE a une propriété spécifique sur le nombre de fois que chaque arc doit être traversé. Lorsqu'un cycle respectant cette propriété est trouvé, DExTaR construit la RTE résultante et la valide par une recherche de son motif dans l'ensemble des reads courts. Pour chaque RTE trouvée, notre algorithme renvoie son motif, le nombre de copies ainsi que les séquences précédant et suivant la RTE. Ces dernières apportent des informations sur l'emplacement de la RTE sur le fragment d'ADN cible.

DExTaR est décrit dans l'Algorithme 10. À noter que l'entrée de l'Algorithme 10 est le graphe de contigs. Cependant, l'algorithme reste le même si, à la place, nous utilisons le graphe d'unitigs comme entrée. Notre algorithme contient trois principales étapes : la détection des cycles (Ligne 1), l'analyse de chaque cycle du point de vue des fréquences de ses arcs (Lignes 4-16) et, dans le cas où une éventuelle RTE est identifiée, la validation de celle-ci en se basant sur les reads courts (Lignes 17-24). Chaque étape est décrite dans les paragraphes suivants.

4.1.1 Détection des cycles

Comme mentionné précédemment, on peut potentiellement retrouver des RTE à partir de chaque cycle élémentaire. Par conséquent, la première étape de DExTaR consiste à détecter les cycles élémentaires du graphe $G^k_c(R)$, en utilisant la méthode $RechercheCycles$.

Les graphes de de Bruijn, de unitigs ou de contigs d'un organisme complexe peuvent être de très grande taille, avec des centaines de milliers de cycles. Afin de détecter un nombre maximum de cycles élémentaires dans un laps de temps limité, nous utilisons l'une des méthodes de détection de cycles les plus efficaces, à savoir l'algorithme de Johnson [53].

L'algorithme de Johnson effectue un parcours en profondeur du graphe à partir de chaque nœud v en retournant les cycles qui contiennent v et qui n'ont pas été encore détectés. La complexité de l'algorithme de Johnson est en $O((N + A) * C)$, où N est le nombre de nœuds, A le nombre d'arcs et C le nombre de cycles élémentaires dans le graphe. Cependant, même avec un algorithme si efficace, la recherche de cycles dans l'intégralité d'un graphe nécessite un temps d'exécution de plusieurs semaines, voire plusieurs mois. Pour limiter la recherche, nous avons introduit le paramètre λ_{max} représentant la longueur maximum en matière de nœuds pour les cycles détectés. Ce paramètre limite la profondeur du parcours du graphe, et donc le nombre de nœuds visités lors de la recherche de cycles.

Algorithme 10 Algorithme de DExTaR

Entrée : L'ensemble R de reads courts, le graphe de contigs $G_c^k(R)$ et un entier λ_{max} qui définit la longueur maximale des cycles recherchés

Sortie : Un ensemble E de RTE identifiées et un ensemble Θ contenant leurs séquences adjacentes dans le fragment d'ADN cible

$C \leftarrow RechercheCycles(G_c^k(R), \lambda_{max})$
$E \leftarrow \emptyset$
$\Theta \leftarrow \emptyset$
Pour chaque $c \in C$ **faire**
  $A_{in}(c) \leftarrow RechercheArcsEntrants(c, G_c^k(R))$
  $A_{out}(c) \leftarrow RechercheArcsSortants(c, G_c^k(R))$
  Pour chaque $\alpha_{in} \in A_{in}(c)$ **faire**
    Pour chaque $\alpha_{out} \in A_{out}(c)$ **faire**
      $F_{arc} \leftarrow CalculFrequences(R, c, A_{in}(c), A_{out}(c), \alpha_{in}, \alpha_{out})$
      Si $ProprieteCycle(F_{arc})$ **alors**
        $\varepsilon \leftarrow AssemblageRTE(c, F_{arc}, \alpha_{in}, \alpha_{out})$
        $E \leftarrow E \cup \{\varepsilon\}$
        $\Theta(\varepsilon) \leftarrow \Theta(\varepsilon) \cup SequencesAdjacentes(\alpha_{in}, \alpha_{out})$
      fin Si
    fin Pour
  fin Pour
  Pour chaque $\varepsilon \in E$ **faire**
    $valid \leftarrow RechercheReads(\varepsilon, R, \Theta(\varepsilon))$
    Si $valid = faux$ **alors**
      $Supprimer(E, \varepsilon)$
      $Supprimer(\Theta, \Theta(\varepsilon))$
    fin Si
  fin Pour
fin Pour
Retourner E, Θ

4.1.2 Analyse des cycles pour la détection des répétitions en tandem exactes

Dans le but de simplifier la présentation de la procédure utilisée pour l'étape d'analyse des cycles, nous allons la décrire en utilisant le graphe de de Bruijn. Néanmoins, dû au processus d'assemblage, la procédure reste la même sur un graphe d'unitigs ou de contigs.

Chaque cycle détecté par l'algorithme de Johnson est analysé afin de trouver une RTE potentielle. Soit c un cycle dans le graphe $G^k(R)$ tel que c est formé par une RTE ε du fragment d'ADN cible D. On considère $v_1 = \varepsilon[1, k]$ et $v_e = Suff(\varepsilon, k)$ respectivement comme les premiers et derniers k-mers de ε comme présenté dans la Propriété 1. Les régions entourant la RTE ε dans D créent deux arcs $\alpha_{in} = (x, v_1)$ et $\alpha_{out} = (v_e, y)$ dans $G^k(R)$, tels que $x[1] + \varepsilon + y[k]$ apparaît dans D. On considère que ces deux arcs marquent le début et la fin de ε.

Dans le cas d'un fragment D complexe (c'est-à-dire contenant beaucoup de répétitions), ε ou des sous-chaînes de ε peuvent apparaître plusieurs fois à différents endroits de D. Par la suite, nous appelons ces séquences des *répétitions dispersées supplémen-*

taires (RDS) de ε. Les RDS et ε partagent des k-mers communs représentés par les mêmes nœuds dans le graphe de de Bruijn. Dans le graphe $G^k(R)$, les chemins correspondant aux RDS traversent donc des parties du cycle c. Cela a deux conséquences pour c. La première est que, comme pour ε, les régions adjacentes des RDS peuvent créer des arcs entrants et sortants supplémentaires à partir des sommets de c. La deuxième conséquence découle du fait que les RDS apparaissent dans des reads de l'ensemble R (comme elles font partie du fragment D). Ainsi, les arcs de la RTE ε en commun avec une RDS ont une fréquence correspondant à la fois à la RTE et à la RDS.

Par conséquent, afin de détecter si un cycle de $G^k(R)$ contient une RTE, nous devons éliminer l'influence des RDS potentielles sur les fréquences des arcs du cycle.

Pour cela, on commence par calculer pour chaque cycle c l'ensemble des arcs entrants et sortants des nœuds de c (Lignes 5-6 dans l'Algorithme 10). Ces ensembles sont définis de la façon suivante. Soient $A_{G_k(R)}$ l'ensemble des arcs de $G_k(R)$, A_c l'ensemble des arcs de c et V_c l'ensemble de nœuds de c. Alors les ensembles contiennent les arcs suivants $A_{in}(c) = \{(x, v) \in A_{G_k(R)}$ tel que $x \notin V_c$ et $v \in V_c\}$ et $A_{out}(c) = \{(v, y) \in A_{G_k(R)}$ tel que $v \in V_c$ et $y \notin V_c\}$.

Malheureusement, les chemins des RTE et des RDS ne sont pas connus. Ainsi, nous supposons que chaque couple $(\alpha_{in}, \alpha_{out}) \in A_{in}(c) \times A_{out}(c)$ est un marqueur potentiel pour le début et la fin d'une RTE ε dans D, correspondant au cycle c. Par conséquent, les arcs dans $A_{in}(c) - \{\alpha_{in}\} \cup A_{out}(c) - \{\alpha_{out}\}$ sont les extrémités des chemins définis par toutes les RDS de ε et leur fréquence doit être enlevée des arcs de c. Cela est réalisé par la méthode $CalculFrequences(R, c, A_{in}(c), A_{out}(c), \alpha_{in}, \alpha_{out})$ qui retourne le tableau F_{arc} contenant les fréquences des arcs de c correspondant uniquement à une RTE potentielle marquée par α_{in} et α_{out}. L'heuristique utilisée dans cette méthode est détaillée dans l'Algorithme 11.

Au départ, la fréquence $F_{arc}[\alpha]$ d'un arc $\alpha \in A_c$ est initialisée avec la valeur de $f(\alpha, R)$ qui est calculée à partir du nombre d'occurrences du (k+1)-mer correspondant dans R. On commence par parcourir le cycle c et, pour chaque nœud $v_i \in V_c$, on calcule les sommes $F_{input}[v_i]$ et $F_{output}[v_i]$ des fréquences des arcs entrants et respectivement des arcs sortants. Il faut noter que ces sommes correspondent uniquement aux fréquences apportées par les RDS. Pour les calculer, on ne prend pas en compte les arcs du cycle ni les arcs définissant la RTE, α_{in} et α_{out}. On utilise ensuite une fréquence f_{acc} qui accumule les fréquences apportées par les RDS. Dans un premier temps, la fréquence f_{acc} est égale à 0. Pour chaque nœud v_i, nous ajoutons à f_{acc} la différence entre $F_{input}[v_i]$ et $F_{output}[v_i]$. Si $f_{acc} > 0$, alors sa valeur représente la fréquence des RDS, et, donc en excès, de l'arc $\alpha_i = (v_i, v_{i+1})$ et on la retire de $F_{arc}[\alpha_i]$. Si sa valeur est négative, cela veut dire que des arcs sortants de v_i correspondent à des RDS pour lesquelles nous n'avons pas encore rencontré les arcs marquant leur début pendant notre parcours du cycle. En conséquence, sa valeur absolue correspond à la fréquence en excès de l'arc précédent v_i, $\alpha_i = (v_{i-1}, v_i)$, et elle est éliminée lors d'un second parcours du cycle. Dans ce cas, $f_{output}[v_i] = |f_{acc}|$ et nous fixons $f_{acc} = 0$. Dans les deux cas, nous fixons $F_{input}[v_i] = 0$ puisque son impact est maintenant inclus dans f_{acc}.

Un exemple du déroulement de l'algorithme est détaillé sur la Figure 4.1.

Le principe de l'heuristique présentée dans l'Algorithme 11 est donc d'accumuler les fréquences des arcs marquant l'entrée des RDS et d'en faire ressortir le maximum dès qu'on rencontre un arc marquant la sortie des RDS. Cela nous permet d'enlever l'impact des RDS malgré le fait que l'on ne connaisse pas la correspondance entre leurs arcs de début et de fin.

Algorithme 11 Algorithme de nettoyage des fréquences des cycles

Entrée : L'ensemble R des reads, le cycle c, les ensembles $A_{in}(c)$ et $A_{out}(c)$ des arcs entrants et sortants des nœuds de c, et les arcs $\alpha_{in} = (x, v_1)$, $\alpha_{in} \in A_{in}(c)$ et $\alpha_{out} = (v_e, y)$, $\alpha_{out} \in A_{out}(c)$

Sortie : Le tableau F_{arc} contenant les fréquences des arcs de c correspondant seulement à la RTE marquée par $(\alpha_{in}, \alpha_{out})$

1: $V_c \leftarrow (v_1, v_2, \ldots, v_m)$
2: $f_{acc} \leftarrow 0$
3: **Pour** $i \leftarrow 1$ **à** m **faire**
4: $\quad \alpha_i \leftarrow (v_i, v_{(i+1) \mod m})$
5: $\quad F_{arc}[\alpha_i] \leftarrow f(\alpha_i, R)$
6: $\quad F_{input}[v_i] \leftarrow \sum_{\substack{q=(x,v_i) \\ q \in A_{in}(c) - \{\alpha_{in}\}}} f(q, R)$
7: $\quad F_{output}[v_i] \leftarrow \sum_{\substack{u=(v_i,y) \\ u \in A_{out}(c) - \{\alpha_{out}\}}} f(u, R)$
8: $\quad f_{acc} \leftarrow max(f_{acc} + F_{input}[v_i] - F_{output}[v_i], 0)$
9: $\quad F_{arc}[\alpha_i] \leftarrow F_{arc}[\alpha_i] - f_{acc}$
10: $\quad F_{output}[v_i] \leftarrow max(-f_{acc}, 0)$
11: $\quad F_{input}[v_i] \leftarrow 0$
12: **fin Pour**
13: **Si** $f_{acc} > 0$ **alors**
14: $\quad$ **Pour** $i \leftarrow 1$ **à** m **faire**
15: $\quad\quad \alpha_i \leftarrow (v_i, v_{(i+1) \mod m})$
16: $\quad\quad f_{acc} \leftarrow max(f_{acc} - F_{output}[v_i], 0)$
17: $\quad\quad F_{arc}[\alpha_i] \leftarrow F_{arc}[\alpha_i] - f_{acc}$
18: $\quad$ **fin Pour**
19: **fin Si**
20: Retourner F_{arc}

Soit le cycle c avec son ensemble de nœuds tel que $c = (v_1, v_2, \ldots, v_n)$. Nous considérons deux RDS dans ce cycle : ψ_1 marquée par les arcs (x_{ψ_1}, v_i) et (v_j, y_{ψ_1}), et ψ_2 marquée par les arcs (x_{ψ_2}, v_s) et (v_t, y_{ψ_2}) tels que $1 < i < s < t < j < n$. Pour simplifier, on note la fréquence apportée par ces deux RDS $f(\psi_1)$ et $f(\psi_2)$. La fréquence accumulée pour les arcs du chemin $(v_i, \ldots, v_s)$ correspond à la fréquence apportée par l'arc (x_{ψ_1}, v_i) et donc à ψ_1. La fréquence accumulée devient donc $f_{acc} = f(\psi_1)$. Pour le chemin $(v_s, \ldots, v_t)$ on rajoute la fréquence apportée par l'arc (x_{ψ_2}, v_i) et donc correspondant à ψ_2. La fréquence accumulée devient alors $f_{acc} = f(\psi_1) + f(\psi_2)$. Une fois arrivé au nœud v_t, on enlève de la fréquence accumulée la fréquence de l'arc (v_t, y_{ψ_2}) et donc de ψ_2. Cela donne $f_{acc} = f(\psi_1) + f(\psi_2) - f(\psi_2) = f(\psi_1)$. La fréquence accumulée restante correspond à la fréquence de ψ_1 qui va être enlevée de la fréquence des arcs du chemin $(v_t, \ldots, v_j)$. Une fois arrivé à v_j dans notre parcours, la fréquence accumulée devient $f_{acc} = f(\psi_1) - f(\psi_1) = 0$, car le chemin de ψ_1 est fini avec l'arc (v_j, y_{ψ_1}).

Il faut noter que cette heuristique a été définie pour le cas où les arcs marquant le début et la fin de la RTE diffèrent de ceux des RDS. Dans le cas où ces arcs coïncident, nous ne pouvons pas déterminer avec certitude la fréquence apportée par les RDS et celle de la RTE.

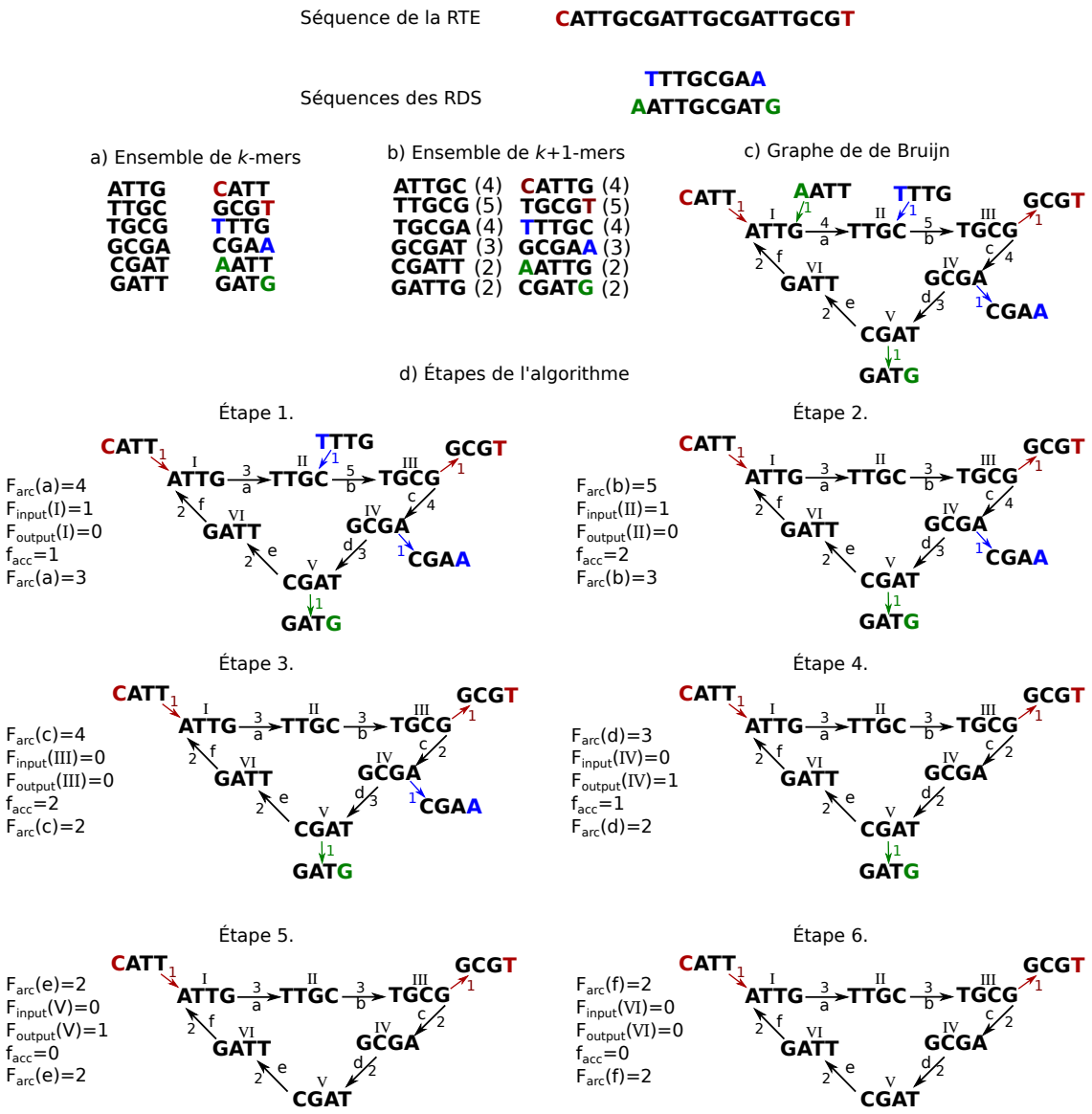


FIGURE 4.1 – Processus de nettoyage de l'impact des RDS sur les fréquences des arcs du cycle représentant une RTE. Dans le graphe de de Bruijn de la RTE présenté sur la Figure 3.9 (k=4) nous avons rajouté deux RDS **TTTGCGAA** et **AATTGCGATG**. Les séquences adjacentes ainsi que les arcs dans le graphe de de Bruijn sont marqués en rouge pour la RTE et en bleu et vert pour les deux RDS. **a)** L'ensemble des k-mers des séquences de la RTE et des RDS. **b)** L'ensemble des (k+1)-mers avec leur nombre d'occurrences dans les séquences de la RTE et des RDS. **c)** Le graphe de de Bruijn initial correspondant aux séquences de la RTE et des RDS. Les nœuds sont numérotés de I à VI et les arcs de a à f. Au départ, la fréquence pour chaque arc du graphe de de Bruijn est donnée par le nombre d'occurrences du (k+1)-mer correspondant dans la RTE et les deux RDS (entre parenthèses au b)). Par souci de simplification, nous choisissons pour la profondeur de couverture $\delta = 1$. **d)** Les étapes de l'algorithme en partant du nœud I et l'arc a. Pour chaque étape nous calculons la fréquence entrante (F_{input}) et sortante (F_{output}) d'un nœud ainsi que la fréquence accumulée (f_{acc}). Nous affichons également les valeurs de la fréquence pour chaque arc F_{arc} avant et après que la fréquence apportée par les RDS a été enlevée.

Pour enlever au moins une partie de la fréquence des RDS, nous dupliquons chacun de ces arcs et nous divisons la fréquence en deux. On suppose qu'un premier arc correspond à la RTE avec une fréquence égale au $min(f(\alpha_{in}, R), f(\alpha_{out}, R))$. La fréquence restante pour l'arc marquant le début $f(\alpha_{in}, R) - min(f(\alpha_{in}, R), f(\alpha_{out}, R))$ et pour l'arc marquant la fin $f(\alpha_{out}, R) - min(\alpha_{in}, \alpha_{out})$ correspond donc aux RDS. Deux arcs sont donc créés avec ces fréquences restantes, l'un étant rajouté à l'ensemble $A_{in} - \{\alpha_{in}\}$, et l'autre à l'ensemble $A_{out} - \{\alpha_{out}\}$.

Un autre cas où notre heuristique ne va pas pouvoir enlever correctement les fréquences des RDS est le cas où des RDS sont plus longues que le cycle. Étant plus longues que le cycle, ces RDS sont donc des RTE qui traversent ν fois les arcs du chemin entre son arc de début et son arc de fin et $\nu - 1$ fois les autres arcs, tel que $\nu > 1$. Ne connaissant pas ce nombre ν, notre heuristique enlève une seule fois la fréquence de la RDS des arcs du chemin entre son arc de début et son arc de fin. La fréquence restante de la RDS correspond donc à $\nu - 1$ pour tous les arcs de notre cycle. En conséquence, les fréquences des arcs du cycle c vérifient quand même la Propriété 1 et nous pouvons ensuite en déduire la RTE. Le seul inconvénient est que la RTE déduite va avoir un nombre de copies supérieur à celle du fragment cible D. Cela est dû au fait que chaque arc va être parcouru $\nu - 1$ fois de plus pour assembler la RTE.

Pour en déduire une RTE potentielle à partir de c, la fréquence F_{arc} restante des arcs de c doit respecter la Propriété 1. Cela est vérifié par la méthode $ProprieteCycle$ (Ligne 10 dans l'Algorithme 10). Dans ce cas, nous assemblons la RTE ε formant le cycle c avec la méthode $AssemblageRTE$ (Ligne 11 dans l'Algorithme 10). Cela est fait en traversant le cycle c en commençant par v_1 et finissant par v_e. Le nombre de fois que chaque arc est parcouru est donné par sa fréquence dans F_{arc}. Les séquences adjacentes s_{avant} et s_{apres} pour la RTE ε sont calculées par la méthode $SequencesAdjacentes$ (Ligne 13 dans l'Algorithme 10) à partir des arcs α_{in} et α_{out}. Cela est fait de la façon suivante. Soit $\alpha_{in} = (x_{in}, v_1)$ alors $s_{avant} = Pref(x_{in}, |x_{in}| - (k-1))$. Respectivement, soit $\alpha_{out} = (v_e, y_{out})$ alors $s_{apres} = Suff(x_{out}, |x_{out}| - (k-1))$. Ces séquences offrent des informations supplémentaires concernant la localisation de la RTE dans le fragment d'ADN cible.

4.1.3 Validation des répétitions en tandem exactes

En raison du fait que les assembleurs basés sur le graphe de de Bruijn découpent les reads en k-mers, une partie de l'information donnée par les reads est perdue. En conséquence, l'ensemble E des RTE obtenues par l'analyse des cycles peut contenir des faux positifs. Par conséquent, la prochaine étape consiste à valider (ou non) chaque RTE $\varepsilon \in E$ en utilisant les reads dans R. Cette étape est réalisée par la méthode $RechercheReads$ (Ligne 18 dans l'Algorithme 10).

Comme présenté dans [15], nous pouvons établir la probabilité de trouver un read dans R couvrant une séquence d'une certaine longueur, étant donné la longueur du fragment d'ADN cible. Nous calculons ensuite le seuil l_{max} pour la taille de la séquence maximum qui peut être trouvée dans au moins un read, avec une forte probabilité.

Soit $\varepsilon \in E$ une RTE détectée pendant la deuxième étape de DExTaR. On considère que ε peut être présente dans le fragment d'ADN cible avec une forte probabilité, s'il y existe des reads dans R attestant les trois conditions suivantes : (i) la présence en tandem du motif p, (ii) la présence de la séquence $s_{avant} + Pref(\varepsilon, l_{max})$ et (iii) la présence de la séquence $Suff(\varepsilon, l_{max}) + s_{apres}$. Nous rappelons que s_{avant} et s_{apres} représentent les

potentielles séquences adjacentes à ε dans le fragment d'ADN cible.

Selon les reads dans R et la relation entre l_{max} et $|\varepsilon|$, ces conditions peuvent être validées avec un seul read qui contient la séquence complète ε comme une sous-chaîne, ou avec plusieurs reads représentatifs.

Si des reads validant la présence de la RTE avec ses séquences adjacentes n'ont pas été trouvés, alors la RTE et ses séquences adjacentes sont supprimées. Sinon on renvoie la RTE avec ses séquences adjacentes.

4.2 Application du programme DExTaR

Cette section est dédiée aux adaptations faites pour nos expérimentations. Nous présentons d'abord l'assembleur de de Bruijn utilisé pour construire le graphe d'unitigs ou de contigs nécessaire pour DExTaR. Ensuite, nous détaillons les modifications apportées à DExTaR pour le cas réel des reads courts erronés ayant une couverture non-homogène. Enfin, nous exposons les mesures de validation utilisées pour nos tests.

4.2.1 Choix de l'assembleur

De nombreux assembleurs de novo sont basés sur un graphe de de Bruijn. Afin de tester DExTaR, nous avons cherché un assembleur qui fournit de bons résultats avec une implémentation du graphe de de Bruijn le plus proche possible de sa définition classique. Suite aux analyses existant dans la littérature [104], ainsi qu'à l'analyse effectuée dans la Section 3.3, nous avons opté pour ABySS [111]. ABySS (Assembly By Short Sequencing) est un assembleur de novo conçu pour reads courts, et basé sur une structure de données de graphe de de Bruijn telle que celle présentée par Pevzner dans [96]. Pour nos expériences, nous utilisons la version 1.3.7 de ABySS.

Afin d'obtenir les données nécessaires pour notre algorithme, nous avons modifié légèrement l'implémentation d'ABySS. En effet, l'assembleur fournit en sortie uniquement les séquences (par exemple unitigs, contigs et scaffolds) pour lesquelles la fréquence moyenne des k-mers les constituant est supérieure à un seuil spécifique. Comme les séquences qui ne satisfont pas ce seuil peuvent représenter des parties des RTE non assemblées, une sortie du graphe complet a été ajoutée après chaque étape d'assemblage. La liste des (k+1)-mers avec leurs fréquences, nécessaire pour notre algorithme, a été obtenue par l'exécution de la procédure de construction des k-mers de ABySS avec la valeur $k + 1$.

4.2.2 Modifications de l'algorithme pour le cas de données réelles

Après avoir analysé le comportement d'ABySS et ses résultats, nous avons inclus quelques modifications dans les méthodes présentées dans l'Algorithme 10. Pour réduire le temps d'exécution, l'assembleur construit un arc entre deux k-mers en vérifiant seulement si leur chevauchement possède une longueur de (k-1) bp. Par conséquent, dans le graphe de de Bruijn résultant, nous pouvons trouver des arcs qui ne sont certifiés par aucun read. Ces arcs provoquent des erreurs lors de l'assemblage. Pour éviter cela, dans notre algorithme, nous considérons uniquement les arcs pour lesquels le nombre d'occurrences du (k + 1)-mer correspondant dans l'ensemble de reads est supérieur à un seuil. Nous avons fixé ce seuil à $\delta/2$, où δ est la couverture de l'ensemble des reads, pour éviter de considérer les arcs construits à cause des erreurs dans les reads. Ainsi, la méthode

$RechercheCycles$ renvoie uniquement les cycles pour lesquels chaque arc vérifie cette condition. De la même manière, la condition est vérifiée aussi par les arc retournés par les méthodes $RechercheArcsEntrants$ et $RechercheArcsSortants$.

En outre, l'Algorithme 10 a été adapté pour le cas réel de reads erronés avec une couverture non-homogène. Soit c, un cycle dans le graphe $G^k(R)$ formé par une RTE ε du fragment d'ADN cible D. Dans le cas réel de couverture non-homogène pour les reads de l'ensemble R, les bases de ε ne sont pas toujours couvertes par le même nombre de reads courts. Ainsi, les fréquences des arcs de c peuvent fluctuer. Par conséquent, la Propriété 1 est vérifiée en utilisant un intervalle plutôt qu'une valeur spécifique pour la fréquence d'un arc. Cet intervalle est calculé en fonction de la couverture de l'ensemble R de reads et de la fréquence moyenne des arcs de c.

4.2.3 Mesures de validation

Pour évaluer l'amélioration apportée par DExTaR pour la détection des RTE, nous considérons d'abord les RTE obtenues avec ABySS et ensuite celles obtenues avec DExTaR.

Le brin d'ADN dont proviennent les unitigs et les contigs assemblés par ABySS n'est pas connu. De plus, le processus d'assemblage peut laisser certaines RTE incomplètes, des préfixes ou suffixes de celles-ci n'apparaissant pas dans les unitigs ou les contigs. En conséquence, pour analyser les résultats obtenus avec ABySS et DExTaR, nous utilisons la notion de *motif atomique*. Étant donné un motif p, le *motif atomique* de p est le premier motif (dans l'ordre lexicographique) obtenu à partir de p par des permutations circulaires de p et de son complément $\overline{p}$ (voir exemple dans la Figure 4.2). Ainsi, plusieurs motifs partagent le même motif atomique : toutes les motifs identiques à une permutation circulaire d'un motif p ou de son complément $\overline{p}$ ont le même motif atomique.

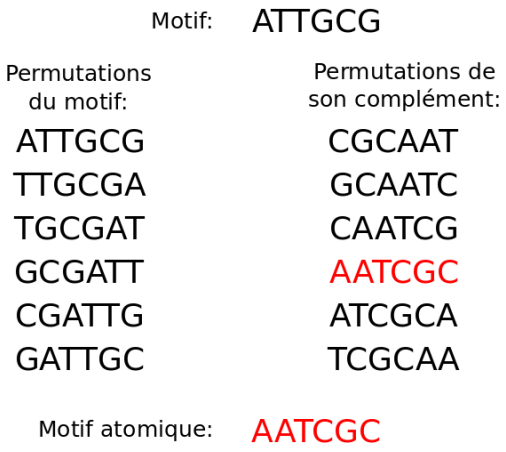


FIGURE 4.2 – Calcul du motif atomique (en rouge) pour le motif *ATTGCG* à partir de ses permutations circulaires et des permutations circulaires de son complément.

Pour chaque RTE détectée, DExTaR récupère aussi ses séquences adjacentes qui fournissent des informations concernant la localisation de la RTE sur le fragment d'ADN cible. Afin d'analyser la qualité de ces informations supplémentaires, nous utilisons la notion de *motif étendu*. Un motif étendu est défini par le triplet $(p, s_{avant}, s_{apres})$, où les séquences s_{avant} et s_{apres} sont celles qui précèdent et suivent une RTE détectée du motif p.

Après l'exécution d'un outil pour la détection des RTE sur les séquences assemblées par ABySS et sur le fragment d'ADN cible D, nous extrayons les motifs atomiques ou étendus. Cela est réalisé sur l'ensemble des RTE identifiées par DExTaR. Ensuite, les résultats obtenus par ABySS ou DExTaR sont analysés en classant chaque motif atomique ou étendu dans une catégorie :

- Vrai Positif (VP) si nous identifions le motif atomique/étendu dans l'ensemble des motifs atomiques/étendus de D ;
- Faux Négatif (FN) si le motif atomique/étendu apparaissant dans D n'est pas détecté dans l'ensemble des motifs atomiques/étendus de nos résultats ;
- Faux Positif (FP) si nous identifions le motif atomique/étendu dans nos résultats mais qu'il n'apparaît pas dans D.

La qualité des résultats obtenus par ABySS et DExTaR est ensuite mesurée en utilisant les deux statistiques suivantes :

- *Précision* $= VP/(VP+FP)$ qui mesure la fraction des motifs atomiques récupérés qui sont présents dans D ;
- *Sensibilité* $= VP/(VP + FN)$ qui mesure la fraction des motifs atomiques de D qui sont correctement identifiés.

4.3 Données utilisées

Pour nos expériences, nous avons besoin d'un organisme complexe présentant de nombreuses RTE. Comme pour les tests réalisés avec les assembleurs de novo (Section 3.3), nous avons choisi *Caenorhabditis elegans*. Le premier chromosome de *C. elegans*, dont la longueur est d'environ de 15Mo, dispose d'une structure très complexe et contient plus de RTE que de nombreux génomes complets.

Nous avons utilisé le même set de reads simulant le modèle d'erreurs d'Illumina que pour les tests effectués sur les assembleurs (Section 3.3), ainsi qu'un deuxième set de reads simulés sans erreurs afin de tester l'impact des erreurs de séquençage. Pour différentier les deux sets de reads dans l'analyse des résultats pour DExTaR, l'ensemble de données simulées sans erreurs est appelé *NER* (No Error Reads), tandis que l'ensemble simulant le modèle d'erreurs d'Illumina est appelé *ER* (Errorneous Reads). Les deux ensembles ont été obtenus avec le simulateur de reads SGS GemSIM [80] et contiennent des reads courts pairés avec une longueur de 100 bp et une couverture de 20x.

Les RTE du première chromosome de *C. elegans* et des séquences assemblées obtenues par ABySS ont été identifiées avec l'outil BWtrs (Burrows-Wheeler Tandem Repeat Searcher) [97]. Comme présenté dans la Section 3.2.2, BWtrs est un logiciel conçu pour la recherche de RTE dans les séquences d'ADN assemblées, les RTE détectées étant maximales avec des motifs primitifs. Les paramètres de BWtrs ont été fixés de telle sorte que la longueur l_p de motifs des RTE retournées satisfait $2 \leq l_p \leq 100$ et $n_p * l_p \geq 4$.

4.4 Analyse des résultats obtenus avec DExTaR

Notre contribution est présentée par la suite en termes de nombre et longueur des motifs atomiques (voir Section 4.2.3) des RTE détectées à partir des graphes d'unitigs/contigs construits par ABySS. La qualité de l'emplacement des RTE est aussi présentée dans une analyse pour les motifs étendus détectés par DExTaR. Comme DExTaR est un outil dédié à l'amélioration de la détection de RTE suite à un assemblage de de Bruijn, nous avons analysé les RTE assemblées par ABySS et le gain obtenu en utilisant DExTaR.

4.4.1 Amélioration obtenue sur le nombre de motifs atomiques détectés

Dans le cas des assembleurs de de Bruijn, la longueur des motifs des RTE assemblées dépend de la valeur du paramètre k [85]. Afin d'analyser cette influence, nous avons exécuté ABySS avec les deux ensembles de reads courts, respectivement avec et sans erreurs, avec $k \in [15, 41]$.

Comme nous l'avons observé auparavant (Section 3.3), la longueur des motifs atomiques des RTE détectées dans les séquences résultantes d'ABySS est toujours inférieure à la valeur de k. En augmentant la valeur de k, ABySS est capable d'assembler des RTE avec des motifs plus longs. Mais l'inconvénient est que le nombre de RTE assemblées diminue de manière significative. En effet, en augmentant le chevauchement nécessaire pour connecter deux k-mers, les RTE courtes sont assemblées de façon erronée.

Dans la Figure 4.3 on peut observer le comportement général d'ABySS. Cette figure présente le nombre de motifs atomiques de RTE détectées pour trois valeurs représentatives de k : 17, 27 et 37. Après l'exécution de BWtrs sur le premier chromosome de *C. elegans*, nous avons extrait à partir des RTE détectées un ensemble contenant 3782 motifs atomiques. Chaque barre représente le nombre de motifs atomiques pour les trois types de motifs atomiques présentés ci-dessus (VP, FP, FN). Le pourcentage de motifs atomiques des RTE détectées par ABySS décroît de plus de 40% en variant k de 17 à 37. Cela est également montré par les valeurs de la sensibilité (voir Tableau 4.1).

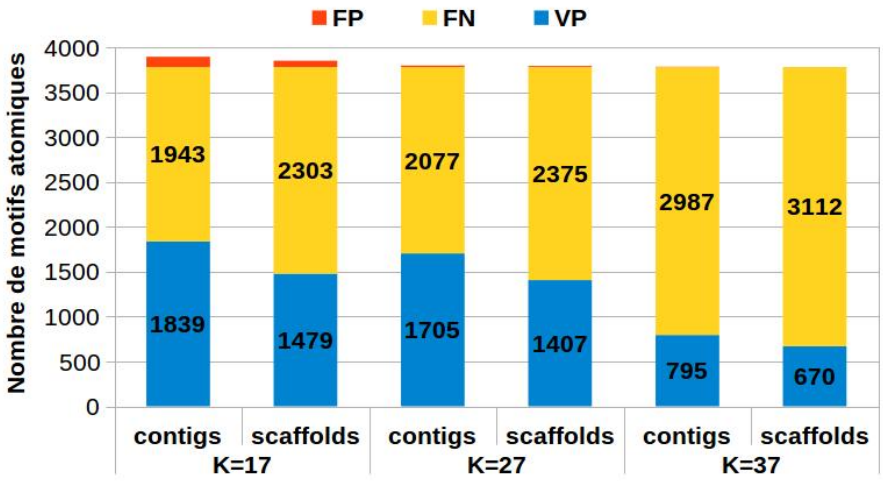


FIGURE 4.3 – Nombre de motifs atomiques des RTE détectées par ABySS pour le premier chromosome de *C. elegans*.

TABLEAU 4.1 – Précision et sensibilité de la détection des motifs atomiques avec ABySS pour le premier chromosome de *C. elegans*.

	Contigs		**Scaffolds**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
K=17	0,941	**0,486**	0,954	**0,391**
K=27	0,990	0,451	0,992	0,372
K=37	**0,996**	0,210	**0,999**	0,177

En outre, nous avons remarqué que pour chaque exécution d'ABySS, l'étape de scaffolding n'assemble pas des RTE avec des nouveaux motifs atomiques par rapport à ceux déjà assemblés dans les contigs. De plus, certains motifs atomiques provenant des RTE correctement assemblées dans les contigs, ne sont plus retrouvés dans les scaffolds.

Mise à part l'influence sur le nombre de motifs atomiques détectés, le paramètre k joue aussi un rôle dans la complexité du graphe d'unitigs ou de contigs. La valeur de k est inversement proportionnelle au nombre de cycles présents dans le graphe représentant potentiellement des RTE laissées non résolues.

Selon les résultats précédents, l'analyse qui suit est basée sur le graphe de contigs obtenu avec k=17. Dans cet ensemble le nombre de RTE assemblées par ABySS est maximal. En raison de la grande importance des procédures de correction d'erreurs pour obtenir un assemblage de qualité et en raison de la faible qualité des unitigs obtenus avec ABySS, nous avons décidé de limiter notre analyse aux contigs.

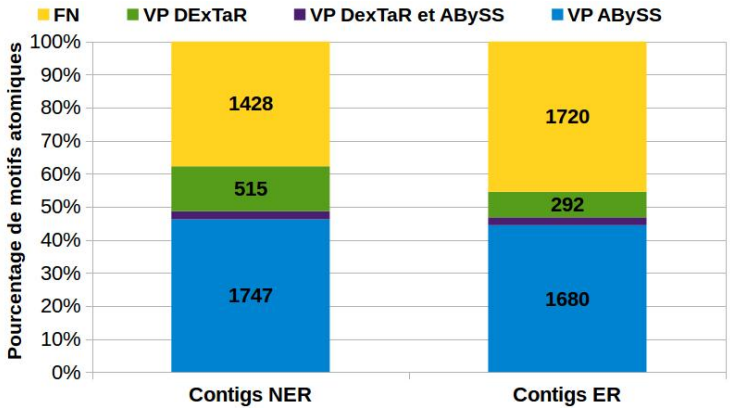
La Figure 4.4a présente l'amélioration apportée par notre algorithme quant au nombre de motifs atomiques détectés du premier chromosome de *C. elegans*. La grande complexité du graphe de contigs construit par ABySS nous oblige à limiter la recherche de cycles à ceux d'une longueur maximale de λ_{max}=15 nœuds.

Néanmoins, DExTaR détecte environ 10 % à 16 % des motifs atomiques du premier chromosome de *C. elegans*. Cela représente une amélioration de 17 % à 29 % par rapport au nombre de motifs atomiques détectés par ABySS. Cette amélioration peut également être observée avec l'augmentation de la sensibilité (voir troisième et cinquième colonnes du Tableau 4.2).

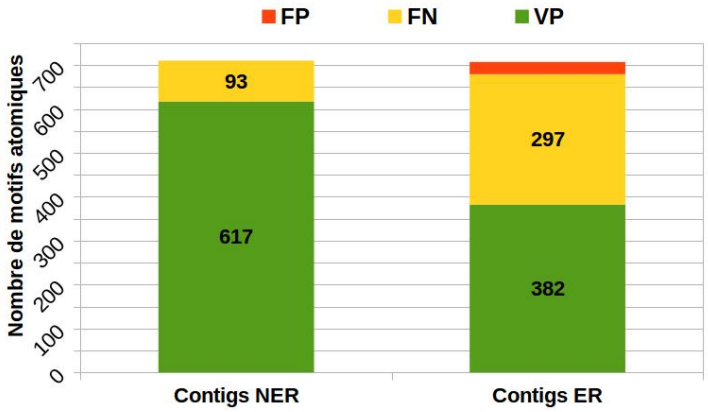
Le fait que le pourcentage de motifs communs détectés à la fois par ABySS et par DExTaR soit inférieur à 2,5 % suggère que ces deux méthodes sont complémentaires. Il faut noter que DExTaR ne recherche pas les RTE dans tout le chromosome, mais uniquement sur ce qui reste de non résolu après l'assemblage des unitigs/contigs par ABySS.

TABLEAU 4.2 – Précision et sensibilité de la détection des motifs atomiques avec ABySS et DExTaR pour le premier chromosome de *C. elegans*.

	ABySS		**ABySS et DExTaR**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
Contigs NER	**0,941**	**0,563**	**0,941**	**0,622**
Contigs ER	0,820	0,507	0,812	0,545



(a) Proportion de motifs atomiques des RTE détectées par ABySS et DExTaR pour le premier chromosome de *C. elegans*



(b) Nombre de motifs atomiques des RTE détectées par DExTaR parmi ceux représentés par les cycles de longueur maximale de 15 nœuds

FIGURE 4.4 – Nombre de motifs atomiques des RTE détectées par ABySS et DExTaR pour le premier chromosome de *C. elegans*

L'analyse détaillée des motifs atomiques détectés par DExTaR par rapport au potentiel des cycles du graphe d'unitigs/contigs dans lequel nous recherchons des RTE est présentée sur la Figure 4.4b. Chaque barre représente l'ensemble des motifs atomiques présents dans les cycles avec au plus 15 nœuds dans le graphe de contigs.

Notre algorithme trouve 86,9 % de cet ensemble de motifs atomiques pour des reads courts simulés sans erreurs (NER). Les faux négatifs sont dus à la non-homogénéité de la couverture des reads courts. Cela crée des écarts élevés entre les fréquences des arcs dans des cycles et DExTaR n'est pas en mesure de déduire correctement les RTE à partir de ces cycles.

Pour l'ensemble des reads simulés avec des erreurs (ER), ce pourcentage décroît à 56,2 %. Une première cause est la variation encore plus importante de la couverture en raison des erreurs dans les reads. L'autre cause est le nombre important de contigs assemblés incorrectement dans le graphe d'ABySS malgré les procédures de correction d'erreurs. Pour ce dernier ensemble de reads, nous avons également détecté environ 7,3 % de faux positifs. Cela entraîne une perte de précision de seulement 0,008 (voir deuxième et quatrième colonnes du Tableau 4.2). La détection de faux positifs est due aux erreurs des reads, chaque motif atomique correspondant à un faux positif détecté en tandem dans les reads mais pas dans le chromosome.

4.4.2 Amélioration obtenue sur la longueur des motifs atomiques détectés

La complémentarité entre ABySS et DExTaR est également confirmée par la longueur des motifs détectés par ces deux outils (voir la Figure 4.5).

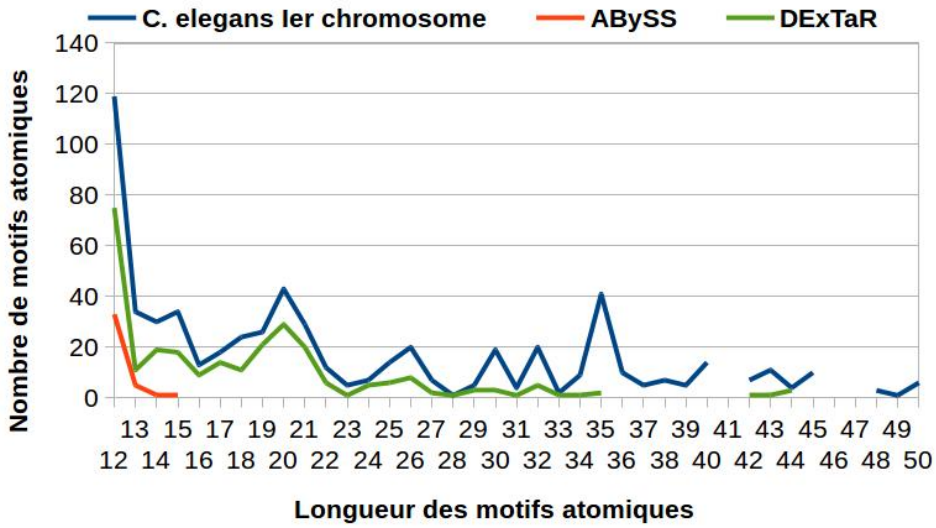


FIGURE 4.5 – Distribution des longueurs des motifs atomiques détectés par DExTaR et ABySS pour le premier chromosome de *C. elegans*

Empiriquement, nous avons observé que les longueurs des motifs atomiques assemblés par ABySS sont limitées par la valeur de k. Pour chaque exécution de l'assembleur, la longueur maximale des motifs des RTE était inférieure à k. Comme présenté dans [116, 120], les microsatellites, les minisatellites et les satellites ont des rôles biologiques essentiels.

Par conséquent, la tâche de DExTaR est d'élargir l'intervalle de longueurs pour les motifs des RTE détectées. Alors que le plus long motif d'une RTE assemblée par ABySS est de 15 bp pour k=17, notre algorithme peut aller jusqu'à 44 bp, comme le montre la Figure 4.5. Le schéma présente le nombre de motifs atomiques avec une longueur $l_p \in [12, 44]$. Les motifs de longueur inférieure à 12 bp représentent 80 % des motifs des RTE présentes dans le premier chromosome de *C. elegans*. Parmi eux, 56,5 % sont découverts par ABySS. En plus de ceux assemblés par ABySS, les motifs atomiques détectés par DExTaR représentent 4,5 %. Cette limitation a deux causes : la première provient des assemblages incorrects dans le graphe d'ABySS, la seconde est que certaines RTE sont trop petites pour être représentées par des cycles.

De plus, la longueur des motifs détectés par notre algorithme est limitée par la longueur des cycles. En augmentant λ_{max}, DExTaR est capable de détecter des motifs atomiques plus longs.

4.4.3 Détection de motifs étendus

Pour chaque RTE détectée, DExTaR est en mesure de la localiser en récupérant les séquences adjacentes. La qualité de cette localisation est indiquée par une analyse des motifs étendus détectés. Comme présenté dans la précédente section, un motif étendu est

défini par le triplet (p, seq_b, seq_e).

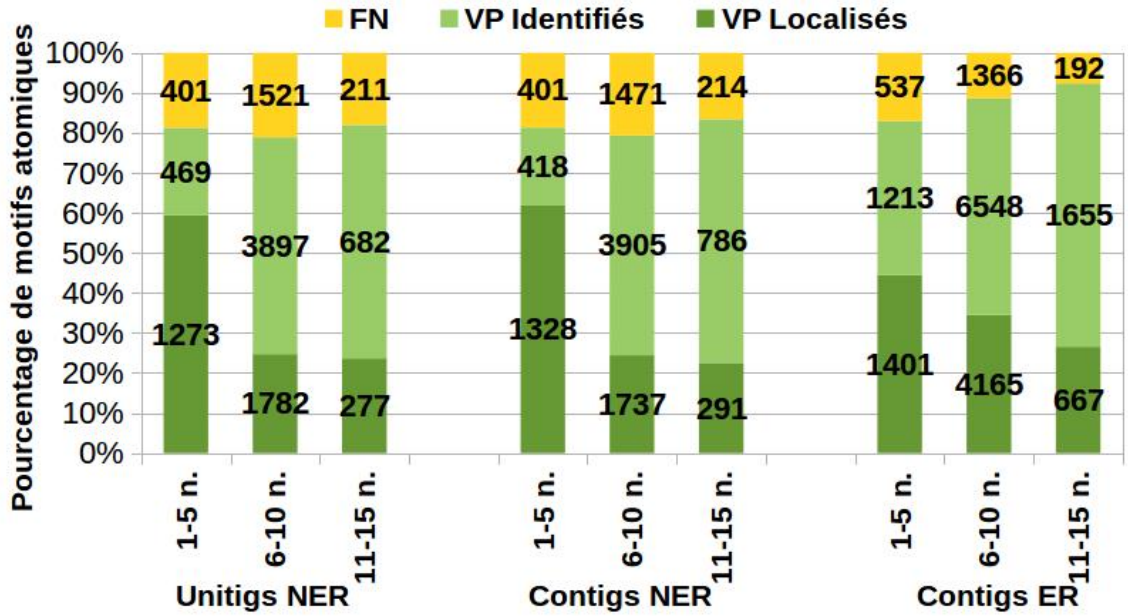


FIGURE 4.6 – Pourcentage de motifs étendus détectés par DExTaR : chaque barre représente le potentiel des cycles de différentes longueurs (de 1 à 5 nœuds, de 6 à 10 nœuds et de 11 à 15 nœuds). Les motifs étendus ont été vérifiés soit avec un seul read (Localisés) ou avec plusieurs reads (Identifiés).

La Figure 4.6 décrit l'ensemble des vrais positifs des motifs étendus détectés par DExTaR. Cette analyse a été effectuée pour des intervalles de longueur de cycles différents. Notre algorithme est capable de détecter de 78,8 % à 92,3 % du potentiel des cycles en ce qui concerne les modèles étendus. La figure expose également la proportion de motifs étendus qui ont été vérifiés en utilisant respectivement un read (*Localisés*), ou plusieurs reads séparés (*Identifiés*). Chaque motif étendu est localisé par DExTaR en identifiant les unitigs/contigs qui précédent et suivent la RTE. Les RTE détectées les plus importantes sont celles qui possèdent une longueur supérieure à celle d'un read (Identifié), car elles ne sont pas présentes en tant que telles dans les reads. La Figure 4.6 montre que le pourcentage de ces motifs étendus (Identifiés) augmente de manière significative avec la longueur des cycles (de 24 % pour les cycles ayant de 1 à 5 nœuds jusqu'à environ 73 % pour les cycles ayant de 11 à 15 nœuds).

4.5 Conclusion

Dans le domaine des données de SGS, les études algorithmiques existantes sont principalement axées sur le problème de l'assemblage global du génome. Les algorithmes d'assemblage de novo de données SGS ont fait beaucoup de progrès récemment, mais l'assemblage d'un génome - et surtout de grands génomes pour des organismes complexes - reste un défi très important. Une des raisons principales est la présence de répétitions pour lesquelles le processus de détection est responsable de nombreuses parties assemblées de façon incorrecte ou tout simplement non assemblées. Afin de proposer une meilleure solution au problème d'assemblage de répétitions, de nombreux algorithmes sont publiés

chaque année, mais la résolution de répétitions dans un assemblage de novo est encore une tâche difficile [117].

Dans le but d'améliorer la détection de répétitions, notre travail a commencé par une analyse approfondie des solutions algorithmiques déjà proposées pour le problème d'assemblage. Après avoir examiné les limites des approches d'assemblage de novo existantes, nous avons proposé un nouvel algorithme qui analyse les parties complexes laissées non assemblées afin de détecter des nouvelles RTE.

En se concentrant sur un type spécifique de répétitions, les RTE, notre approche améliore la qualité de la détection des répétitions en récupérant des RTE complexes non résolues par les assembleurs. La méthode est basée sur le graphe de de Bruijn en raison de son efficacité pour l'assemblage des reads courts, ainsi que sa structure fragmentée qui permet la représentation des RTE sous forme d'un modèle topologique facilement identifiable.

La qualité des résultats est prouvée par la forte exploitation du potentiel des cycles sur lesquels nous recherchons des RTE, ainsi que par la précision des motifs détectés.

5

Algorithme hybride de détection des répétitions en tandem

La méthode DExTaR a pour objectif l'amélioration de la détection des RTE suite à un assemblage global de de Bruijn, comme nous l'avons présenté dans le chapitre précédent. Notre travail de recherche a été étendu aux RTA (Répétitions en Tandem Approximatives) avec une nouvelle méthode appelée MixTaR. MixTaR représente la première méthode de novo pour la détection de répétitions en tandem (RT) qui utilise à la fois les reads courts et les reads longs en effectuant uniquement des assemblages locaux. Le nouvel algorithme hybride combine la bonne qualité des reads courts et la grande longueur des reads longs pour une détection efficace des RT. En détectant à la fois les RTE et les RTA, MixTaR va plus loin que DExTaR dans la détection de RT. En outre, contrairement à DExTaR, MixTaR n'a pas besoin d'un assemblage global préalable.

Ce chapitre présente en détail notre algorithme MixTaR ainsi qu'une analyse de la qualité de ses résultats. Nous commençons par détailler les étapes principales de MixTaR. Ensuite, nous évaluons la qualité des reads longs et de son amélioration apportée par les outils de correction. Enfin, les résultats obtenus avec MixTaR sur des données simulées et réelles sont analysés en fonction des différents taux d'erreurs pour les deux ensembles de reads, courts et longs. Ces travaux ont été réalisés en collaboration avec Guillaume Fertin, Géraldine Jean et Irena Rusu, et ont été publiés dans [36].

5.1 Description détaillée de MixTaR

MixTaR représente une solution au Problème de DÉTECTION DE NOVO HYBRIDE DE RÉPÉTITIONS EN TANDEM qui est définie comme suit :

DÉTECTION DE NOVO HYBRIDE DE RÉPÉTITIONS EN TANDEM

Entrée : Un ensemble *SR* de reads courts et un ensemble *LR* de reads longs, tous deux obtenus suite à un processus de séquençage d'un fragment d'ADN cible D.

Objectif : Trouver l'ensemble des RT du fragment D impliquant les reads de *SR* et *LR* ainsi que leurs compléments.

La haute qualité des reads courts de l'ensemble *SR* et la grande longueur des reads longs de l'ensemble *LR* sont exploitées par MixTaR à travers trois étapes principales (comme présenté sur la Figure 5.1) : (1) la détection des motifs, (2) la validation des motifs et (3) l'assemblage des RT.

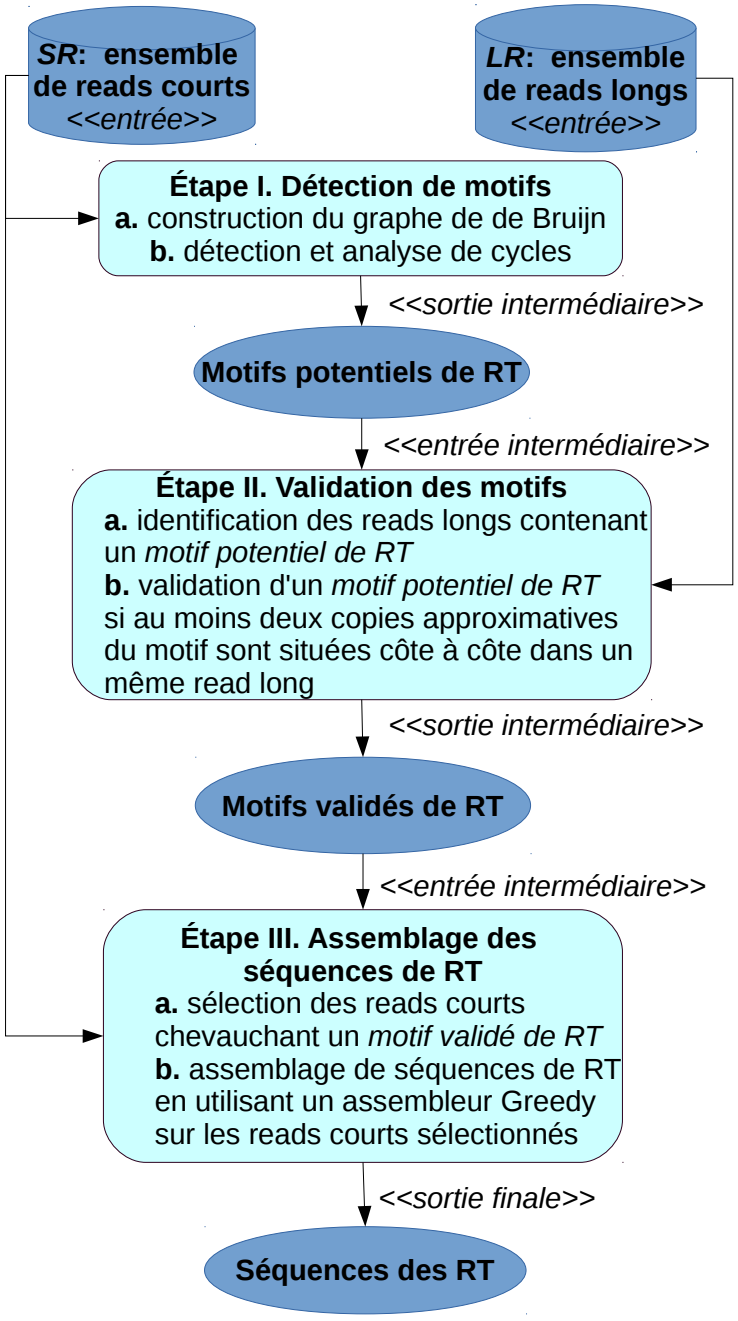


FIGURE 5.1 – Pipeline de MixTaR

Dans la première étape, une fois les compléments des reads courts ajoutés à l'ensemble *SR*, nous construisons le graphe de de Bruijn correspondant. Comme nous l'avons présenté dans la Section 3.4, les RTE forment des cycles dans le graphe de de Bruijn. Ainsi, nous parcourons le graphe pour détecter et analyser les cycles pour ensuite déduire une liste de motifs de RTE potentielles. Il faut noter que contrairement à DExTaR, nous n'enregistrons que les motifs des RTE.

Pour la deuxième étape, nous ajoutons à l'ensemble *LR* les compléments des reads longs et nous utilisons cet ensemble pour valider les motifs des RT potentielles. En raison de la longueur importante des reads longs, la plupart des RT de D sont contenues dans au moins un read de l'ensemble *LR*. La principale difficulté de l'utilisation des reads longs provient de leur taux d'erreurs. Ainsi, nous commençons par chercher des copies approximatives de chaque motif potentiel de RT dans chaque read de *LR*. Nous validons ensuite un motif si nous identifions au moins deux copies approximatives de ce motif situées à côté l'une de l'autre dans au moins un read de *LR*.

La troisième et dernière étape consiste à identifier la séquence exacte des RT contenant les motifs validés. Pour réaliser cela, nous utilisons à nouveau l'ensemble *SR*. Les reads courts chevauchant un motif validé sont utilisés pour un assemblage Greedy local, à partir duquel on obtient la séquence précise des RT contenant le motif.

Chaque étape est détaillée dans les paragraphes qui suivent.

5.1.1 Détection des motifs

Pour la première étape, nous construisons le graphe de de Bruijn correspondant à l'ensemble des reads courts. Ensuite, nous nous basons sur les mêmes propriétés de ce graphe que pour DExTaR en ce qui concerne la représentation des RTE sous la forme de cycles. Ainsi, MixTaR parcourt le graphe de de Bruijn pour détecter les cycles et analyse les cycles identifiés afin d'en déduire la liste des motifs des RTE potentielles.

L'ensemble *SR* contient des reads courts obtenus après le séquençage du fragment d'ADN cible D, ainsi que leur compléments. Comme mentionné précédemment (Section 3.4), les RTE de D avec une longueur d'au moins $|p|+k$ (où p est le motif de la RTE) forment, dans un graphe de de Bruijn $G^k(SR)$, des cycles élémentaires respectant la Propriété 1. Ces RTE peuvent représenter des sous-chaînes des RTA ou RTE plus longues du même motif p qui apparaissent dans D. Dans ce cas, des copies approximatives ou exactes de p sont situées à côté de la RTE dans D et la RTE est considérée comme *interne* à une RT plus longue. Dans ce qui suit, ces RT qui contiennent une RTE interne formant un cycle dans le graphe de de Bruijn sont nommées *RT robustes*. Notre algorithme MixTaR recherche d'abord les motifs des RTE internes, pour ensuite rechercher les RT robustes les contenant. Cela est réalisé à partir de la liste des motifs des RTE potentielles obtenue pendant cette étape.

Détection de cycles

Nous considérons, comme pour DExTaR, que chaque cycle élémentaire représente une RTE potentielle de D. Par conséquent, après la construction du graphe de de Bruijn $G^k(SR)$ pour une valeur spécifique de k, on commence la recherche des cycles dans ce graphe.

Les erreurs de séquençage de l'ensemble *SR* peuvent introduire des nœuds erronés dans $G^k(SR)$. Afin d'éliminer ces nœuds, nous considérons dans notre recherche seule-

ment ceux pour lesquels $occ(v, SR) \geq \sigma$, où σ est un paramètre dont la valeur est calculée en fonction de δ, la couverture de *SR*. Nous rappelons que $occ(v, SR)$ représente le nombre d'occurrences de v dans les reads de SR (voir Section 3.4). En outre, nous trions la liste des nœuds dans $G^k(SR)$ dans l'ordre décroissant de leur nombre d'occurrences dans *SR*.

Afin de détecter un nombre maximum de cycles élémentaires dans un laps de temps limité, nous utilisons comme pour DExTaR, l'algorithme de Johnson [53]. Nous rappelons que l'algorithme de Johnson explore le graphe à partir de chaque nœud v et renvoie les cycles qui contiennent v et qui n'ont pas encore été détectés. Pour limiter cette exploration, nous introduisons trois paramètres : η, Λ_{max} et λ_{max}. Pour chaque nœud v, nous commençons par chercher les cycles ayant une longueur maximale de Λ_{max} nœuds. Après avoir exploré η arcs à partir de v et s'il y a encore des arcs à explorer, l'algorithme recherche les cycles restants à partir de v d'une longueur maximale de λ_{max} ($\lambda_{max} \leq \Lambda_{max}$).

Comme mentionné précédemment, après avoir analysé un cycle c de l nœuds, nous pouvons obtenir un motif p de longueur $2 \leq |p| \leq l$. Par conséquent, afin d'obtenir des motifs de longueur importante, nous devons maximiser le nombre de cycles détectés d'une longueur maximale de Λ_{max} qui contiennent potentiellement une RTE. Pour réaliser cela, nous considérons que les arcs des cycles contenant une RTE ont une fréquence élevée dans *SR*. Ainsi, à partir de chaque nœud, nous commençons par explorer ses arcs sortant dans l'ordre décroissant de leur fréquence dans *SR*. De cette façon, nous traversons les arcs ayant la fréquence la plus élevée dans la première partie de la recherche de cycles qui permet de détecter des cycles de longueur allant jusqu'à Λ_{max} nœuds. Les paramètres Λ_{max}, η et λ_{max} sont fixés en fonction de la complexité du graphe $G^k(SR)$ et de la durée autorisée par l'utilisateur pour l'exécution de la recherche de cycles.

Chaque nœud v à partir duquel nous commençons l'exploration du graphe doit satisfaire une condition supplémentaire : $occ(v, LR) > 0$. En raison de l'ordre dans lequel les nœuds sont considérés, le nœud v, à partir duquel l'algorithme recherche des cycles, est aussi le nœud avec le plus grand nombre d'occurrences dans *SR* parmi les nœuds des cycles obtenus pour v. Nous considérons donc que, parmi les nœuds des cycles détectés à partir de v, nous avons pour v la plus grande probabilité de trouver des occurrences sans erreurs dans *LR*. Cette condition supplémentaire est utilisée dans la deuxième étape de MixTaR, pour valider les motifs obtenus à l'aide de l'ensemble *LR*.

Analyse de cycles

Une fois que les cycles contenant potentiellement des RTE ont été détectés, nous devons les analyser afin d'extraire les motifs de ces RTE. Pour réaliser cela, nous rencontrons le même problème que pour DExTaR : les répétitions dispersées supplémentaires (RDS) des RTE (Section 4.1.2). Une fois de plus, pour chaque cycle c nous calculons l'ensemble des arcs entrants et sortants des nœuds de c (Ligne 4-5 dans l'Algorithme 10) : $A_{in}(c) = \{(x, v) \text{ avec } x \notin V_c \text{ et } v \in V_c\}$ et $A_{out}(c) = \{(v, y) \text{ avec } v \in V_c \text{ et } y \notin V_c\}$. Ensuite, nous supposons que chaque couple $(\alpha_{in}, \alpha_{out}) \in A_{in}(c) \text{x} A_{out}(c)$ est un marqueur potentiel pour le début et la fin d'une RTE ε dans D, correspondant au cycle c. Enfin, l'impact des RDS est supprimé en utilisant l'Algorithme 11.

Pour en déduire un motif de RTE potentielle à partir de c, la fréquence restante des arcs de c doit respecter la Propriété 1. Dans ce cas, nous construisons la RTE ε du motif p formant le cycle c de la manière suivante : soit l le nombre de nœuds de c, alors $\varepsilon = v_1 + v_2[k] + \ldots + v_l[k]$. Comme mentionné précédemment, $l = |p|$ et donc $p = \varepsilon[1, l]$.

5.1.2 Validation des motifs

L'analyse des cycles que nous venons de décrire ne peut retrouver que des motifs potentiels de RTE, et donc, de RT robustes. Cela est dû au fait qu'une partie de l'information donnée par les reads courts est perdue en raison de leur découpage en k-mers. En conséquence, l'ensemble des motifs obtenus à partir du graphe de de Bruijn peut contenir des faux positifs. Ainsi, pour chaque motif obtenu, nous devons valider sa présence dans au moins une RT du fragment d'ADN cible D. Pour cela, nous utilisons l'ensemble *LR* en raison de la longueur importante de ses reads.

Avec une longueur variable qui peut aller de 1 kbp à 20 kbp [16], les reads dans *LR* peuvent couvrir la plupart des RT dans D. L'hypothèse que nous faisons est plus faible : nous supposons que chaque motif p d'une RT de D a au moins deux copies adjacentes contenues dans au moins un read de *LR*. Ainsi, nous validons un motif si nous identifions au moins deux copies de celui-ci situées l'une à côté de l'autre dans au moins un read long. Sinon, le motif est considéré comme un faux positif et il est rejeté.

La difficulté soulevée par cette approche est d'identifier correctement deux copies adjacentes d'un motif en dépit du taux d'erreurs élevé des reads longs, qui est d'environ 16 %. Une solution possible est l'utilisation d'une procédure de correction des reads longs [4, 103, 64, 45]. Cependant, dans le cas des génomes complexes, même les méthodes les plus efficaces ne sont pas en mesure de corriger toutes les erreurs des reads longs (voir Section 5.2). Ainsi, notre méthode de recherche de motifs permet l'utilisation de reads longs corrigés et également de reads longs non corrigés.

Afin de valider un motif p, nous commençons par rechercher une copie approximative de p dans les reads de *LR*. Soit c un cycle détecté à l'étape précédente, et soit v le nœud de c ayant le plus grand nombre d'occurrences dans *SR*. Comme indiqué à l'étape précédente, v satisfait la condition $occ(v, LR) > 0$. Soit p le motif potentiel déduit par l'analyse de c, c'est-à-dire le motif de la RTE formant le cycle c.

Si $|p| < k$ ou si $v = Suff(p, x) + Pref(p, k - x)$ pour un entier $1 \leq x < k$, alors v apparaît dans une séquence obtenue par des concaténations successives de p. Sinon, v apparaît directement dans p. Pour une validation rapide du motif p, on limite la recherche des copies de p aux reads longs contenant v.

Soit s une séquence initialement identique à p. Si $occ(v, s) = 0$, on étend s par des concaténations successives de p de telle sorte que s est la chaîne la plus courte pour laquelle $occ(v, s) > 0$. Le processus est fini comme v apparaît dans la RTE de p formant le cycle c.

Ensuite nous recherchons des occurrences de s dans les reads longs. Pour cela, nous utilisons les notions qui suivent. Comme présenté dans la Section 3.2, le *score d'alignement* entre deux séquences s_1 et s_2 est obtenu à partir d'un alignement de s_1 et s_2 en utilisant une fonction de poids pour chaque opération, insertion, suppression et substitution, ainsi que équivalence entre les caractères. Dans notre cas, nous additionnons 1 pour chaque position identique et -1 pour chaque insertion, suppression et substitution. Le *score maximum d'alignement semi-global* entre s_1 et s_2 est noté $sgA_{max}(s_1, s_2)$ et représente le plus grand score d'alignement obtenu à partir de tous les alignements possibles entre s_1 et une sous-séquence de s_2.

Soit pos une position dans un read long r telle que $r[pos, k] = v$. Nous essayons ensuite d'identifier une occurrence de s dans r en recherchant l'alignement avec le score le plus élevé entre s et une sous-chaîne de r autour de la position pos. Approximativement 70% des erreurs dans les reads de *LR* sont des insertions [19]. Ainsi, nous considérons pour notre alignement une sous-chaîne de r de longueur $2|s|$ et nous calculons le score

maximum d'alignement semi-global $t = sgA_{max}(s, r[pos - |s|, 2|s|])$. Dans le cas où l'occurrence de v dans r est trop proche d'une extrémité de r, alors nous calculons l'alignement soit avec le préfixe de r de longueur $2|s|$ soit avec le suffixe de r le plus long possible commençant à la position $pos - |s|$.

Nous utilisons ensuite un paramètre τ, qui représente le seuil pour notre score t. La valeur de τ est fixée selon le pourcentage approximatif d'erreurs des reads longs que nous utilisons, corrigés ou non corrigés. Si $t/|s| < \tau$, alors p est considéré comme un faux positif. Sinon, soit $occ(p + p, s) > 0$ et le motif p est validé, soit nous répétons le processus pour $s = s + p$.

5.1.3 Assemblage des séquences de répétitions en tandem

La troisième et dernière étape consiste à identifier la séquence exacte des RT robustes contenant les motifs validés. Pour cela, nous utilisons à nouveau l'ensemble des reads courts *SR*. Les reads courts chevauchant un motif validé sont utilisés dans un assemblage local Greedy à partir duquel on obtient les séquences précises des RT robustes contenant le motif.

En raison du taux élevé d'erreurs des reads longs, les RT partielles détectées dans la deuxième étape de MixTaR contiennent une quantité importante de bases erronées. Ainsi, pour la dernière étape de notre algorithme, nous devons trouver la séquence exacte pour chaque RT. Les RT de chaque motif p validé sont construites par un assemblage local de l'ensemble SR_p des reads courts chevauchant p.

Pour construire l'ensemble SR_p pour chaque motif p validé, nous introduisons deux paramètres γ et ϑ.

Plus la longueur du motif p est élevée, plus la probabilité qu'un read dans *SR* chevauche p est également élevée. Par conséquent, la taille de SR_p et le temps d'exécution de son assemblage peuvent être importants pour un motif p court. Soit s une chaîne initialement identique à p. Afin de limiter le nombre de reads courts r utilisés dans les assemblages locaux, si $|s| < \gamma$, nous étendons s par concaténations successives de p tel que s est la séquence la plus courte pour laquelle $|s| \geq \gamma$. La valeur du paramètre γ dépend de la longueur des reads courts dans *SR*, et doit être fixée afin de limiter la taille de SR_p, tout en maximisant le nombre de reads inclus dans SR_p et qui couvrent une partie d'une RTE de p.

Nous construisons alors l'ensemble SR_p en recherchant les reads courts chevauchant s en utilisant la notion suivante. Le *score maximum de l'alignement chevauchant* entre deux séquences s_1 et s_2 est noté $oA_{max}(s_1, s_2, l_{min})$ et représente le plus grand score d'alignement obtenu à partir de tous les alignements possibles de $Pref(s_1, l)$ et $Suff(s_2, l)$, pour $l \geq l_{min}$. MixTaR permet des chevauchements approximatifs car p peut être un motif d'une RTA. Pour chaque read court $r \in SR$, nous calculons le score de l'alignement chevauchant $t = max(oA_{max}(s, r, \gamma), oA_{max}(r, s, \gamma))$. Si $t \geq \vartheta$, alors r est rajouté à SR_p. La valeur de ϑ dépend du taux d'erreurs des reads dans *SR* et de la différence autorisée entre un motif d'une RTA et ses copies.

Une fois l'ensemble SR_p calculé, nous utilisons un assembleur Greedy pour l'assembler. Ce choix a été fait en raison de son temps de calcul réduit pour obtenir des résultats satisfaisants [85]. Nous rappelons qu'un assembleur Greedy calcule une fonction de similarité qui lui est propre pour le chevauchement entre chaque paire de reads courts. Puis les reads courts sont assemblés dans des séquences plus grandes, appelées contigs, par des assemblages successifs de reads courts avec le score de chevauchement le plus élevé.

Dans chaque contig résultant de l'assemblage Greedy, nous pouvons trouver plusieurs RT, et nous devons identifier leurs positions dans le contig. Pour ce faire, nous utilisons un outil de détection des RT dans des séquences connues (voir Section 3.2) qui nous fournit les positions des RT dans chaque contig. L'ensemble des contigs est ensuite analysé. Dans certains cas, nous pouvons obtenir des contigs qui ne contiennent aucune RT. Étant donné que ces contigs sont pas significatifs pour notre détection de RT, ils sont supprimés. Dans le cas où une RT est positionnée à une extrémité d'un contig, on peut supposer que cette RT est incomplète. Le contig devient alors une *graine*. Chaque graine est ensuite étendue pour obtenir la séquence complète de la RT. L'extension des graines est réalisée en utilisant à nouveau l'assembleur Greedy sur l'ensemble de reads $SR - \cup_{p \in \Im} SR_p$, où $\Im$ est l'ensemble de tous les motifs validés au cours de la deuxième étape du MixTaR. Lorsque que toutes les graines sont étendues, nous utilisons l'outil des détection de RT dans des séquences connues pour une identification complète des RT dans l'ensemble des graines étendues. À la fin, nous renvoyons les RT découvertes dans les graines et celles des contigs.

5.2 Qualité des reads longs

La technologie PacBio produit des reads d'une longueur variable allant de 1 kbp à 20 kbp. Malheureusement, l'extension de la longueur des reads s'accompagne d'une augmentation significative du taux d'erreurs, qui est d'environ 16% [19]. La difficulté soulevée par l'utilisation des reads longs est d'identifier correctement deux copies voisines d'un motif en dépit de ce taux d'erreurs élevé. Une solution possible est l'utilisation d'une procédure de correction des reads longs [4, 103, 64, 45]. Cependant, dans le cas de génomes complexes, même les méthodes les plus efficaces ne sont pas en mesure de corriger toutes les erreurs des reads longs. Pour illustrer cela, nous avons corrigé des reads longs simulés et réels avec l'un des outils de correction les plus efficaces, LoRDEC [103]. Les résultats obtenus du point de vue de la qualité de l'alignement des reads longs corrigés sur le fragment d'ADN cible montrent que la correction des reads longs reste un problème difficile surtout, dans le cas de fragments d'ADN cible complexes.

MixTaR est conçu pour fonctionner à la fois avec des reads longs corrigés et non corrigés comme entrée. Pour nos expériences, nous avons utilisé des reads longs simulés avec PBSIM [91] (sur le premier chromosome de *C. elegans* et sur la souche *Philadelphia* de *L. pneumophila*) ou obtenus avec les technologies de séquençage PacBio (pour la souche *130b* de *L. pneumophila*). Pour une analyse complète de l'impact des erreurs des reads longs sur les résultats obtenus avec MixTaR, nous avons corrigé ces ensembles de reads longs avec LoRDEC [103]. LoRDEC est un outil de correction d'erreur pour les reads longs basé sur un graphe de de Bruijn construit à partir d'un ensemble de reads courts. Le paramètre principal pour LoRDEC est k, la longueur des k-mers dans le graphe de de Bruijn. Nous avons exécuté LoRDEC avec des valeurs impaires pour $k \in [15, 31]$ et nous avons aligné les reads longs corrigés sur les séquences d'ADN de référence correspondantes avec la méthode utilisée par GAGE [104]. Le pourcentage de bases alignées de reads longs corrigés sur les séquences d'ADN de référence correspondantes est présenté sur la Figure 5.2. À chaque fois, les meilleurs résultats ont été obtenus avec $k = 19$, comme pour les résultats présentés dans l'article décrivant LoRDEC [103].

Même pour la valeur optimale de k, nous pouvons remarquer que, pour le premier chromosome de *C. elegans*, GAGE n'a pas été en mesure d'aligner environ 30 % des bases de reads longs corrigés sur la séquence de référence [GenBank : GCA_000002985.3].

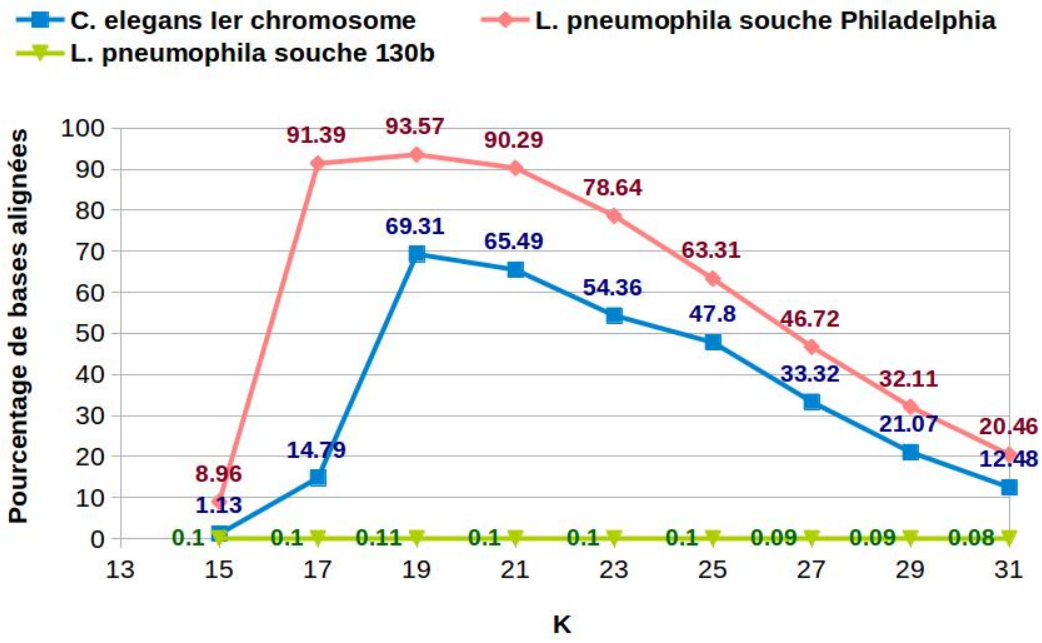


FIGURE 5.2 – Qualité de la correction de reads longs évaluée avec le pourcentage de bases alignées de reads longs corrigés avec LoRDEC sur les séquences d'ADN de référence.

Les valeurs présentées sur la Figure 5.2 sont obtenues pour des reads longs avec une couverture de 20x. Pour des reads longs avec une couverture de 100x, nous avons obtenu des valeurs similaires. Ce pourcentage élevé de bases non alignées est dû à la structure complexe du chromosome dont la longueur est d'environ 15 Mb.

Dans le cas de la souche *Philadelphie* de *L. pneumophila*, le pourcentage de bases alignées est nettement plus élevé. Cela est dû à un ensemble plus réduit de répétitions et à la petite longueur (3,4 Mb) du génome [GenBank : GCA_000008485.1], permettant ainsi une correction de reads longs plus précise. Comme pour le premier chromosome de *C. elegans*, des résultats similaires ont été obtenus pour des reads longs avec des couvertures de 20x et 100x. Sur la Figure 5.2, nous présentons les résultats obtenus pour l'ensemble des reads longs avec une couverture de 20x.

Pour la souche *130b* de *L. pneumophila*, nous avons corrigé un ensemble de reads longs réels [SRA : ERX620205]. Les reads longs corrigés ont ensuite été alignés sur le génome de référence constitué de 159 contigs [GenBank : GCA_000211115.2]. Le faible pourcentage de bases alignées des reads longs corrigés sur le génome encore à l'état de brouillon est probablement causé par le pourcentage d'erreurs dans l'ensemble des reads longs réels qui est plus élevé que les 16 % annoncés [19].

Nous avons également corrigé les trois ensembles de reads avec deux autres outils de corrections, proovread [45] et LSC [4]. Le pourcentage, calculé avec GAGE, de bases alignées de reads longs corrigés sur les séquences d'ADN de référence correspondantes est présenté sur le Tableau 5.1.

Comme LoRDEC, les deux outils de correction se basent sur un ensemble de reads courts pour corriger les reads longs, mais ils ne sont pas paramétrables. Les valeurs présentées dans le Tableau 5.1 ont été obtenues avec les mêmes ensembles de reads courts qu'avec LoRDEC et les ensembles de reads longs avec une couverture de 20 x pour *C.*

TABLEAU 5.1 – Qualité de la correction de reads longs évaluée avec le pourcentage de bases alignées de reads longs corrigés avec proovread et LSC sur les séquences d'ADN de référence ("-" signifie qu'aucun read long n'a été renvoyé par l'outil au bout de 4 semaines).

	Ier chromosome de C. elegans	**L. pneumophila souche Philadelphia**	**L. pneumophila souche 130b**
proovread	-	90,74%	0,00%
LSC	26,68%	50,29%	0,02%

elegans et la souche Philadelphia du *L. pneumophila*, ainsi que l'ensemble de reads longs réels pour la souche 130b du *L. pneumophila*. Nous pouvons observer que les reads longs résultants sont de moins bonne qualité que ceux obtenus par LoRDEC. De plus, les deux outils de correction nécessitent des temps d'exécution plus longs que LoRDEC. Dans le cas de proovread, pour le premier chromosome du *C. elegans*, aucun read long corrigé n'a été renvoyé au bout de 4 semaines d'exécution.

5.3 Application du programme MixTaR

Dans cette section, nous détaillons les adaptations faites pour nos expérimentations. Nous présentons tout d'abord les adaptations apportées à MixTaR pour le cas réel d'une couverture non-homogène pour les ensembles de reads ainsi que les outils et les bibliothèques utilisés pour les expériences. Ensuite, nous exposons les mesures de validation employées pour évaluer la qualité des résultats.

5.3.1 Configuration expérimentale

Pour la première étape de MixTaR (Étape I. de la Figure 5.1), nous utilisons la bibliothèque dédiée à la construction et à l'utilisation des graphes de de Bruijn GATB-lib [29]. Dans cette étape, les résultats des calculs impliquant les fréquences d'arcs présentés dans l'Algorithme 11 dépendent de l'homogénéité de la couverture utilisée pour obtenir l'ensemble *SR*. Soit c un cycle dans le graphe $G^k(SR)$ formé par une RTE ε du fragment d'ADN cible D. Dans le cas réel de couverture non-homogène de l'ensemble *SR*, les bases de ε ne sont pas toujours couvertes par le même nombre de reads courts. Ainsi, les fréquences des arcs de c peuvent fluctuer. Par conséquent, la Propriété 1 est vérifiée, comme dans le cas de DExTaR en utilisant un intervalle plutôt qu'une valeur spécifique pour la fréquence d'un arc. Cet intervalle est calculé en fonction de la couverture de l'ensemble *SR* et de la fréquence moyenne des arcs en c.

Dans l'étape de validation des motifs de MixTaR (Étape II. de la Figure 5.1), nous calculons le score maximum d'alignement semi-global entre les motifs et les reads longs en utilisant les méthodes d'alignement proposées par la bibliothèque SeqAn [28]. Nous avons également utilisé cette bibliothèque à la fin de notre algorithme (Étape III. de la Figure 5.1) pour calculer les chevauchements entre les motifs et les reads courts. Pour effectuer les assemblages locaux Greedy, nous avons choisi SSAKE [123] en raison de son temps d'exécution réduit et de sa configuration simple pour obtenir des résultats satisfaisants [132].

Pour nos expériences, nous utilisons l'outil de recherche de RT pour des séquences assemblées appelé mreps [60] afin d'identifier les RT des contigs obtenus avec SSAKE. Nous rappelons que mreps est un logiciel conçu pour identifier rapidement des RTE et des RTA avec des motifs primitifs dans les séquences d'ADN. Nous avons utilisé mreps pour identifier également les RT dans les séquences d'ADN de référence des organismes testés. En raison du nombre important de RT dans les organismes testés, mreps a été paramétré à chaque fois pour trouver les RT avec au moins deux copies complètes.

Pour chaque étape de MixTaR, les principaux paramètres qui doivent être fixés par l'utilisateur sont les suivants :

- Étape I : Détection des motifs
  - la longueur k des k-mers pour la construction du graphe de de Bruijn.
  - le nombre minimum d'occurrences σ d'un k-mer dans *SR*. La valeur de ce paramètre dépend de la couverture de *SR*.
  - la longueur maximale Λ_{max} des cycles détectés à partir de chaque nœud v.
  - le nombre maximal η d'arcs explorés à partir de chaque nœud du graphe pour la recherche de cycles de longueur maximale Λ_{max}. Les valeurs de Λ_{max} et de η dépendent de la taille du graphe de de Bruijn et de la durée d'exécution autorisée par l'utilisateur.
  - la longueur maximale λ_{max} de cycles détectés pour un nœud v du graphe une fois que η, le nombre maximal d'arcs visités à partir de v, a été atteint. La valeur λ_{max} ($\lambda_{max} \leq \Lambda_{max}$) dépend également de la taille du graphe de de Bruijn et de la durée d'exécution autorisée par l'utilisateur.
- Étape II : Validation des motifs
  - le score minimum τ autorisé pour l'alignement entre un motif et un read long. La valeur de ce paramètre dépend du taux d'erreurs des reads longs.
- Étape III : Assemblage des séquences de RT
  - la longueur minimale γ du motif utilisé pour sélectionner les reads courts pour les assemblages locaux. Pour nos expériences, les meilleurs résultats pour des reads courts d'une longueur de 100 bp ont été obtenus avec $\gamma = 10$.
  - le score minimum ϑ autorisé pour l'alignement entre un motif et un read court, et également entre le motif et ses copies. Pour nos expériences, les meilleurs résultats ont été obtenus avec $\vartheta = 10$.

5.3.2 Mesures de validation

Le brin d'ADN des contigs obtenus avec SSAKE n'étant pas connu, nous ne connaissons donc pas le brin des RT détectées. C'est pour cela que nous considérons, quand nous détectons une RT et son complément, qu'ils représentent la même RT. Ainsi, une RT détectée peut être représentée soit par sa séquence soit simultanément par sa séquence et son complément. Dans notre analyse, nous considérons seulement le brin de référence du fragment d'ADN cible D. Ainsi, une RT est correctement identifiée si nous détectons la séquence (seule ou avec son complément).

En conséquence, chaque RT est classée comme suit :

- Vrai Positif (VP) si nous détectons la séquence complète (c'est-à-dire avec le nombre correct de copies) de la RT de D ;
- Vrai Positif incomplet (VP_i) si nous détectons seulement une partie de la RT de D ;
- Faux Négatif (FN) si nous ne détectons pas la RT de D ;
- Faux Positif (FP) si nous détectons la RT, mais que sa séquence n'apparaît pas dans D.

La qualité des résultats obtenus par MixTaR est ensuite mesurée en utilisant les deux statistiques suivantes :

- *Précision* $= (VP + VP_i)/(VP + VP_i + FP)$ qui mesure la fraction des RT récupérées qui sont présentes dans D ;
- *Sensibilité* $= (VP + VP_i)/(VP + VP_i + FN)$ qui mesure la fraction des RT de D qui sont correctement identifiées.

5.4 Analyse des résultats obtenus avec MixTaR

Dans cette section, nous présentons les résultats obtenus avec MixTaR pour deux types d'ensembles de données : simulées et réelles. Comme mentionné précédemment, notre algorithme est construit pour analyser tous les cycles de longueur maximale λ_{max} du graphe de de Bruijn et seulement une partie des cycles de longueur comprise entre λ_{max} et Λ_{max}. De plus, la longueur d'un motif déduit à partir d'un cycle du graphe de de Bruijn est égale au nombre de nœuds du cycle. Par conséquent, pour les RT détectées avec MixTaR nous présentons deux analyses distinctes. Dans un premier temps, nous décrivons les résultats obtenus sur les RT avec une longueur maximale du motif de λ_{max} bp. Puis nous étendons notre analyse aux RT avec une longueur maximale du motif de Λ_{max} bp. Ces analyses sont effectuées à la fois sur l'ensemble des RT robustes (celles représentant la cible de notre algorithme) et sur l'ensemble des RT générales appartenant aux séquences de référence. Une partie des RT générales ne possèdent pas de RTE internes formant un cycle dans le graphe de de Bruijn et ne sont donc pas directement visées. Elles sont néanmoins détectées par MixTaR.

5.4.1 Données simulées utilisées

Pour tester MixTaR, nous avons, encore une fois, choisi le premier chromosome de *C. elegans* [GenBank : GCA_000002985.3]. Ce chromosome, d'une longueur d'environ 15 Mb, possède une structure très complexe et contient plus de RT que de nombreux génomes complets. Nous avons exécuté mreps [60] pour identifier les RT de sa séquence de référence et nous avons obtenu un ensemble de 39 006 RT avec le motif p d'une longueur $2 \leq |p| \leq 100$.

Pour les reads courts nous avons utilisés les deux mêmes ensembles de reads courts pairés utilisés pour l'analyse de la qualité de DExTaR (Section 4.3). Nous rappelons que les reads courts ont été obtenus en utilisant le simulateur GemSIM [80] et ont une longueur de 100 bp et une couverture de 20x. Pour différencier ces ensembles de reads courts des ceux de reads longs, l'ensemble des données sans erreurs est appelé *SR-NE* (Short Reads with No Errors), tandis que l'ensemble simulant le modèle d'erreurs d'Illumina

est appelé *SR-E* (Short Reads with Errors). Les deux ensembles sont utilisés dans nos tests pour analyser l'impact des erreurs de séquençage des reads courts sur la qualité des résultats de MixTaR. Deux autres ensembles de reads courts pairés ont été obtenus en corrigeant l'ensemble *SR-E* : le premier (*SR-CE1*, pour Short Reads with Corrected Errors 1) en utilisant l'outil de trimming proposé par SSAKE [123] et le second (*SR-CE2* pour Short Reads with Corrected Errors 2) en utilisant l'outil de correction proposé par ALL-PATHS [14], qui est l'une des méthodes de correction d'erreurs les plus efficaces pour les reads courts [104].

Des ensembles de reads longs avec une couverture de 20x et 100x ont été obtenus en utilisant le simulateur des reads longs PacBio PBSIM [91] (*LR-E 20x et 100x* - Long Read with Errors). Deux autres ensembles de reads longs ont été obtenus en corrigeant *LR-E 20x et 100x* avec LoRDEC (*LR-CE* 20x et 100x - Long Reads with Corrected Errors).

5.4.2 Analyse des résultats obtenus avec les données simulées

Pour les expériences sur le premier chromosome de *C. elegans*, nous avons utilisé les valeurs suivantes pour les paramètres de MixTaR. Comme les reads courts ont une longueur de 100 bp, nous avons testé toutes les valeurs impaires de $k \in [17, 47]$ pour la construction du graphe de de Bruijn. En raison de la couverture des reads courts (20x), nous avons appliqué l'algorithme de recherche de cycles pour tous les k-mers ayant un nombre minimal d'occurrences $\sigma = 10$. Pour le premier chromosome de *C. elegans*, le graphe de de Bruijn présente un nombre important de cycles. Pour la recherche de cycles, nous avons donc utilisé $\eta = 10.000$ arcs pour les cycles de longueur maximale $\Lambda_{max} = 100$ et $\lambda_{max} = 20$. Pour le score minimal d'alignement τ, nous avons utilisé la valeur $\tau = 10$ pour les reads longs non corrigés et $\tau = 20$ pour ceux corrigés.

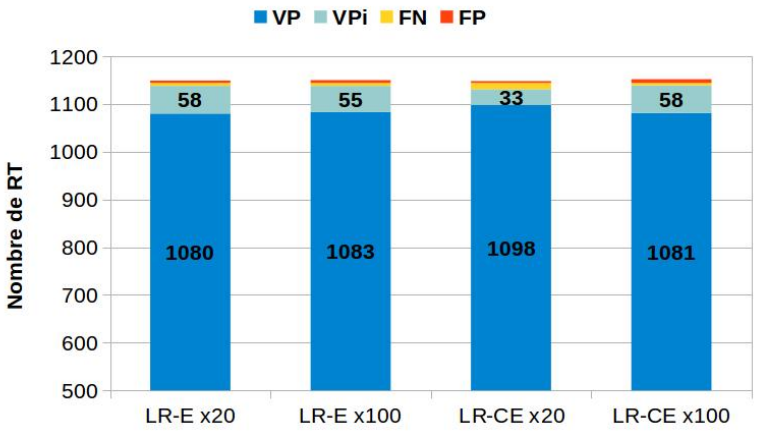
Dans les paragraphes suivants, les résultats obtenus avec MixTaR sont analysés à partir de deux points de vue : le pourcentage de RT détectées à partir du premier chromosome de *C. elegans* et la longueur de leurs motifs.

Pourcentage de RT détectées en fonction de la qualité des reads longs

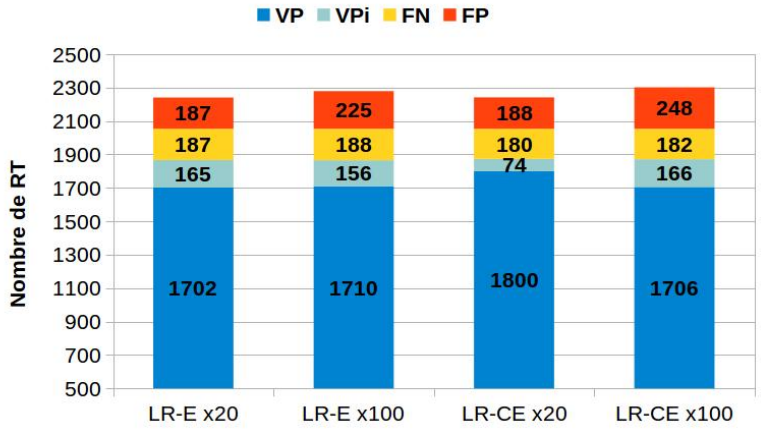
Afin d'évaluer l'impact des erreurs des reads longs sur la qualité des résultats obtenus par MixTaR, nous exécutons MixTaR sur les quatre ensembles de reads longs décrits dans le paragraphe précédent, en l'occurrence *LR-E* et *LR-CE* avec une couverture de 20x et 100x. Afin d'évaluer uniquement l'impact des erreurs des reads longs, nous nous plaçons dans le cas de conditions optimales pour ces derniers. Ainsi, pour ces tests, nous avons utilisé l'ensemble sans erreurs pour les reads courts, *SR-NE*, et différentes valeurs impaires pour $k \in [17, 47]$. Dans ce qui suit, nous présentons les résultats obtenus pour $k = 17$, la valeur pour laquelle MixTaR renvoie le plus grand ensemble de RT.

La Figure 5.3 décrit les résultats obtenus pour l'ensemble des RT robustes. Après l'exécution de mreps sur le premier chromosome de *C. elegans*, nous avons extrait les RT robustes à partir de l'ensemble des RT retournées par mreps. Pour $k = 17$, nous avons obtenu 1144 RT robustes avec $|p| \leq 20$ et 2054 RT robustes avec $|p| \leq 100$. Chaque barre de la Figure 5.3 représente le nombre de RT robustes pour les quatre types de RT présentés ci-dessus (VP, VP_i, FP et FN).

MixTaR est capable de détecter la séquence complète de plus de 94 % des RT robustes avec $|p| \leq 20$ du chromosome. Le pourcentage décroît à 82,8 % pour les RT robustes avec $|p| \leq 100$, en raison du fait que MixTaR ne peut explorer tous les cycles d'une longueur



(a) Nombre de RT robustes avec une longueur de motif maximale de 20 bp



(b) Nombre de RT robustes avec une longueur de motif maximale de 100 bp

FIGURE 5.3 – Nombre de RT robustes détectées pour le premier chromosome de *C. elegans* utilisant des reads longs avec différents taux d'erreurs. Les résultats sont obtenus avec les 4 ensembles de reads longs simulés non corrigés *LR-E* et corrigés *LR-CE*, avec une couverture de 20x et 100x. Pour chaque essai, nous avons utilisé l'ensemble *SR-NE* de reads courts et $k = 17$.

comprise entre 20 et 100 nœuds. Ce pourcentage pourrait être augmenté en permettant une durée d'exécution plus grande et une valeur plus élevée pour le paramètre η (c'est-à-dire le nombre maximal d'arcs explorés pour un nœud afin de trouver les cycles de longueur maximale de 100 nœuds). En outre, le pourcentage des FP est d'environ 0,3 % pour les RT avec $|p| \leq 20$ et d'au plus 12 % pour les RT avec $|p| \leq 100$. Les FP obtenus sont dus aux assemblages locaux Greedy. Même si l'ensemble des reads courts est sans erreurs, SSAKE peut faire de mauvais choix lors de l'assemblage de l'ensemble des reads pour un motif de RT. Cela se produit lorsque, en raison de longues répétitions, deux reads ont le plus long chevauchement mais ne sont pas situés dans la même région du chromosome. Cependant, en sélectionnant l'ensemble adéquat de reads courts pour les assemblages locaux Greedy, nous sommes en mesure de limiter le nombre de FP obtenus, contrairement à un assemblage Greedy global.

Les valeurs exactes pour la précision et la sensibilité obtenues sur l'ensemble des RT

TABLEAU 5.2 – Précision et sensibilité de la détection des RT robustes et des RT générales avec comme longueur maximale du motif 100 bp pour le premier chromosome de *C. elegans* et pour différents ensembles de reads longs. Les résultats sont obtenus avec l'ensemble *SR-NE* de reads courts et $k = 17$. En gras, les meilleures valeurs obtenues pour chaque catégorie de RT.

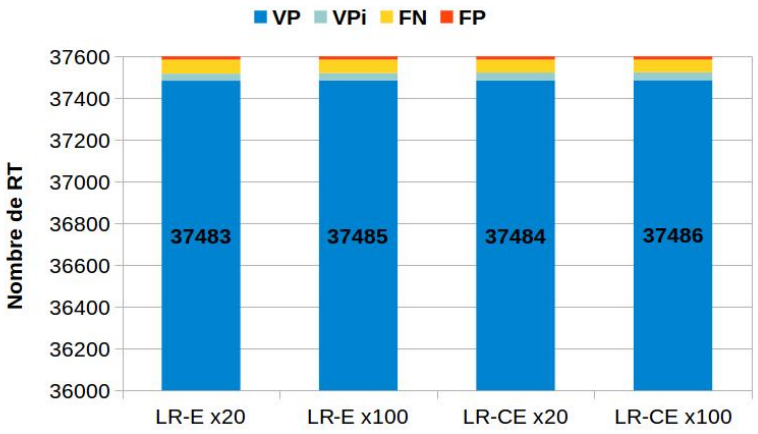
	RT robustes		**RT générales**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
LR-E 20x	**0.909**	0.909	0.990	**0.984**
LR-E 100x	0.892	0.908	0.988	**0.984**
LR-CE 20x	**0.909**	**0.912**	**0.993**	**0.984**
LR-CE 100x	0.883	0.911	0.987	**0.984**

robustes avec $|p| \leq 100$ sont présentées dans les deux premières colonnes du Tableau 5.2. Il n'y a pas de différence notable entre les ensembles de RT obtenus avec les quatre ensembles de reads longs. Ceci est probablement dû au fait que, dans le cas d'un organisme complexe avec nombre important de répétitions, la correction des reads longs reste une tâche difficile. En utilisant les reads longs uniquement pour la validation des motifs de RT, MixTaR est fiable par rapport au taux d'erreurs des reads longs.

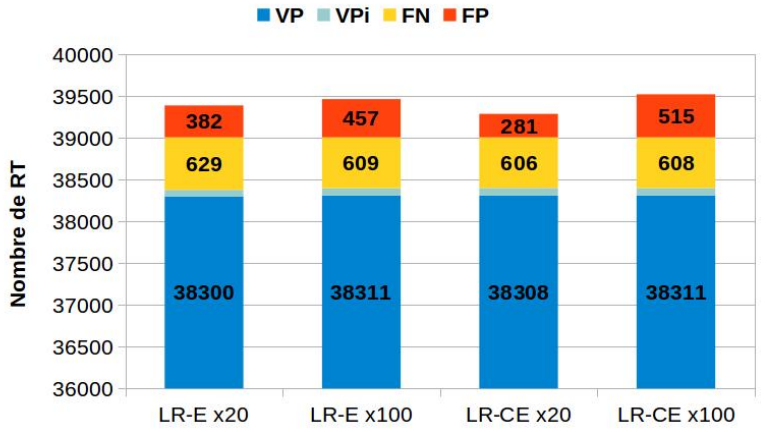
Outre les RT robustes, MixTaR détecte également des RT générales tel que présenté sur la Figure 5.4. Une des raisons principales à la détection des RT générales réside dans le fait qu'un grand nombre de RT du premier chromosome de *C. elegans* ont leur motif très similaire à au moins un motif d'une RT robuste. Dans ce cas, en sélectionnant tous les reads courts chevauchant un motif d'une RT robuste, nous sommes également en mesure d'assembler des RT générales de ce motif ou d'un motif similaire. Une autre raison est l'emplacement des RT générales. En effet, en assemblant les séquences des RT robustes, on obtient également les régions adjacentes. Ainsi MixTaR peut identifier les RT générales qui se trouvent dans ces régions.

À partir des résultats de mreps sur le premier chromosome de *C. elegans* on obtient 37584 RT avec $|p| \leq 20$ et 39006 RT avec $|p| \leq 100$. Les résultats de MixTaR sont meilleurs sur ces ensembles de RT générales, avec plus de 98 % de RT détectées et moins de 1,3 % de FP. Cela est dû au fait que le premier chromosome de *C. elegans* contient un nombre important de RT avec des motifs similaires. En sélectionnant les reads courts les plus significatifs pour les assemblages locaux, nous sommes en mesure de détecter un pourcentage élevé de RT générales sans introduire d'erreurs.

Les valeurs de la précision et de la sensibilité sur l'ensemble des RT générales avec $|p| \leq 100$ sont données sur la troisième et la quatrième colonne du Tableau 5.2. Une fois de plus, on observe, dans l'ensemble, que la correction des reads longs n'a pas de véritable influence sur la qualité des résultats de MixTaR. Nous pouvons cependant remarquer que le nombre de FP obtenus pour les RT avec la longueur maximale du motif de 100 bp est le plus faible avec l'ensemble *LR-CE* 20x qu'avec les trois autres ensembles (voir Figure 5.4). Mais cela ne représente qu'environ 0,6 % de toutes les RT détectées par MixTaR. Comme le coût de correction des reads longs quant au temps d'exécution et à l'utilisation de la mémoire est important, dans ce qui suit, nous présentons les résultats obtenus en utilisant l'ensemble des reads longs non corrigés *LR-E* avec la plus faible couverture, 20x.



(a) Nombre de RT générales avec une longueur de motif maximale de 20 bp



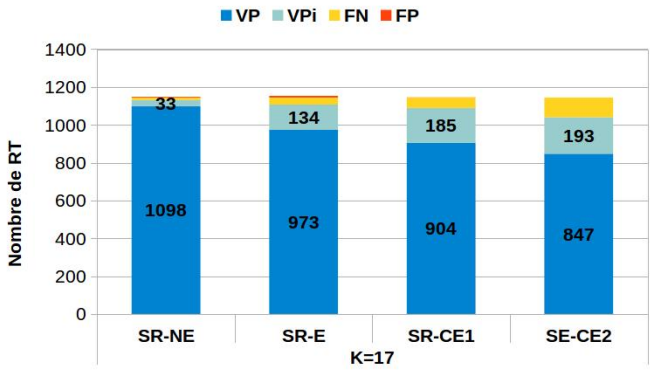
(b) Nombre de RT générales avec une longueur de motif maximale de 100 bp

FIGURE 5.4 – Nombre de RT générales détectées pour le premier chromosome de *C. elegans* utilisant des reads longs avec différents taux d'erreurs. Les résultats sont obtenus avec les 4 ensembles de reads longs simulés non corrigés *LR-E* et corrigés *LR-CE*, avec une couverture de 20x et 100x. Pour chaque essai, nous avons utilisé l'ensemble *SR-NE* des reads courts et k=17.

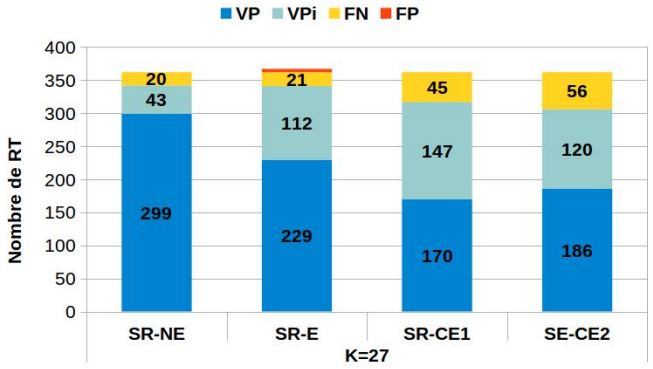
Pourcentage de RT détectées en fonction de la qualité des reads courts et de la variance de la valeur de k

La robustesse de MixTaR par rapport au taux d'erreurs dans les reads courts est analysée en utilisant les quatre ensembles de reads courts décrits dans le paragraphe précédent : *SR-NE*, *SR-E* et les deux ensembles de reads courts corrigés, *SR-CE1* et *SR-CE2*. L'ensemble de reads longs utilisé est *LR-E* avec une couverture de 20x. Pour chaque ensemble de reads courts, nous avons exécuté plusieurs fois MixTaR avec différentes valeurs impaires de $k \in [17, 47]$. Dans ce paragraphe, nous présentons les résultats obtenus pour quatre valeurs significatives de k, à savoir 17, 27, 37 et 47.

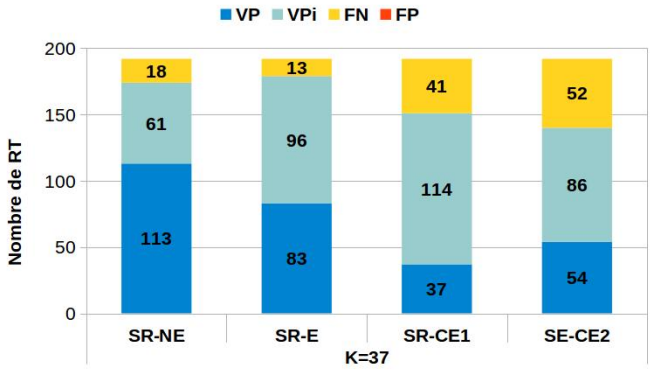
Comme précédemment, nous commençons notre analyse par l'ensemble des RT robustes. La Figure 5.5 présente les résultats obtenus pour les RT robustes pour lesquelles $|p| \leq 20$. Le nombre de RT robustes varie en fonction de la valeur de k que nous utilisons.



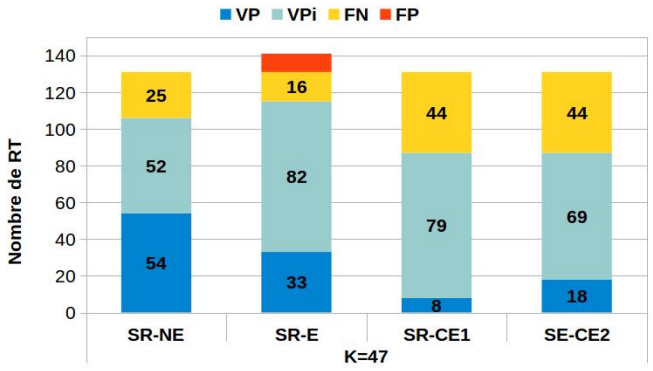
(a) Nombre de RT robustes avec une longueur de motif maximale de 20 bp obtenues pour k=17



(b) Nombre de RT robustes avec une longueur de motif maximale de 20 bp obtenues pour k=27



(c) Nombre de RT robustes avec une longueur de motif maximale de 20 bp obtenues pour k=37



(d) Nombre de RT robustes avec une longueur de motif maximale de 20 bp obtenues pour k=47

FIGURE 5.5 – Nombre de RT robustes détectées avec une longueur maximale de motif de 20 bp pour le premier chromosome de *C. elegans*. Les résultats sont obtenus avec l'ensemble *LR-E* des reads longs avec une couverture de 20x.

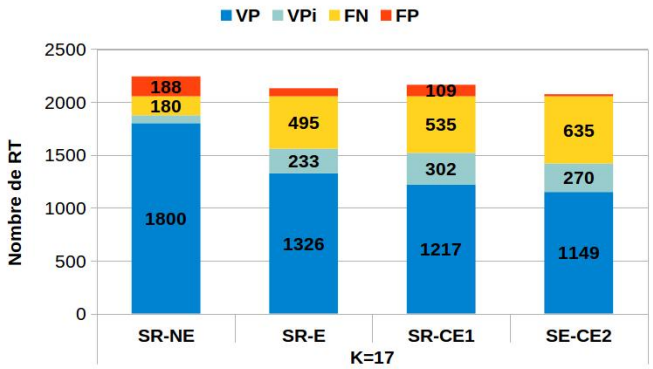
Après l'exécution de mreps sur le premier chromosome de *C. elegans*, nous avons trouvé 1114 RT robustes pour k = 17, 362 RT robustes pour k = 27, 192 RT robustes pour k = 37 et le nombre est réduit à 131 RT robustes pour k = 47. Chaque barre de la Figure 5.5 représente l'ensemble des RT robustes pour la valeur correspondante de k ainsi que l'ensemble des FP renvoyés par MixTaR. Les meilleurs résultats sont obtenus pour k = 17, ce qui peut être expliqué par le fait que plus la valeur de k est réduite, plus les reads courts sont fragmentés. Par conséquent, le nombre de RT formant un cycle dans le graphe de de Bruijn augmente, et notre algorithme a alors la possibilité de les détecter. En outre, la forte fragmentation des reads courts obtenus avec de petites valeurs de k permet une meilleure identification de la variation des fréquences des arcs provoquée par la non-homogénéité de la couverture. Ainsi MixTaR est en mesure de mieux identifier le cas où un motif de RT peut être déduit à partir d'un cycle.

Les résultats obtenus pour les RT robustes avec $|p| \leq 100$ sont exposés Figure 5.6. Sur le premier chromosome de *C. elegans*, nous avons obtenu avec mreps 2054 RT robustes pour k = 17, 1093 RT robustes pour k = 27, 712 RT robustes pour k = 37 et 558 RT robustes pour k = 47. Nous observons, une fois de plus, que la qualité des résultats augmente lorsque l'on diminue la valeur de k. La principale raison est celle mentionnée dans le paragraphe précédent : une fragmentation des reads courts plus importante crée plus de cycles et implique un meilleur calcul de la fréquence des arcs par MixTaR. En outre, la correction de l'ensemble des reads courts *SR-E* peut réduire le nombre de FP obtenus de plus de 90 %. Étonnamment, avec les ensembles de reads courts (*SR-CE1* et *SR-CE2*) le nombre de VP peut également diminuer, jusqu'à 63 %. Ceci est probablement dû aux méthodes de correction d'erreurs pour les reads courts, qui sont trop strictes sur la qualité des bases des reads courts.

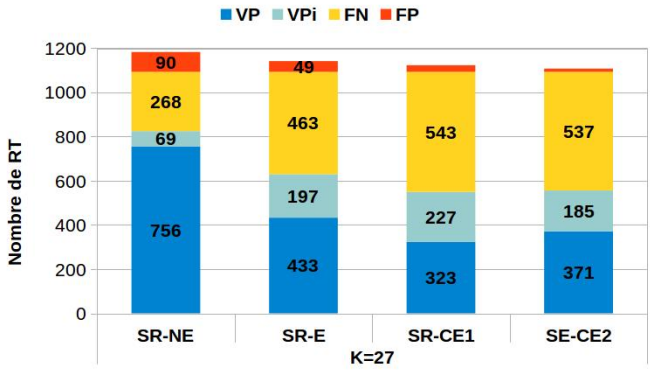
TABLEAU 5.3 – Précision et sensibilité de la détection des RT générales avec comme longueur maximale de motif 100 bp pour le premier chromosome de *C. elegans* et pour différents ensembles de reads courts. Les résultats sont obtenus avec l'ensemble *LR-E 20x* des reads longs. En gras, les meilleures valeurs obtenues pour chaque valeur de k.

	K=17		**K=27**		**K=37**		**K=47**	
	Prec.	**Sens.**	**Prec.**	**Sens.**	**Prec.**	**Sens.**	**Prec.**	**Sens.**
SR-NE	0.990	**0.984**	0.994	**0.980**	0.996	**0.948**	0.997	**0.884**
SR-E	0.872	0.917	0.942	0.862	0.961	0.758	0.968	0.638
SR-CE1	0.984	0.939	0.994	0.906	0.997	0.777	0.997	0.579
SE-CE2	**0.998**	0.818	**0.999**	0.764	**0.999**	0.669	**0.999**	0.565

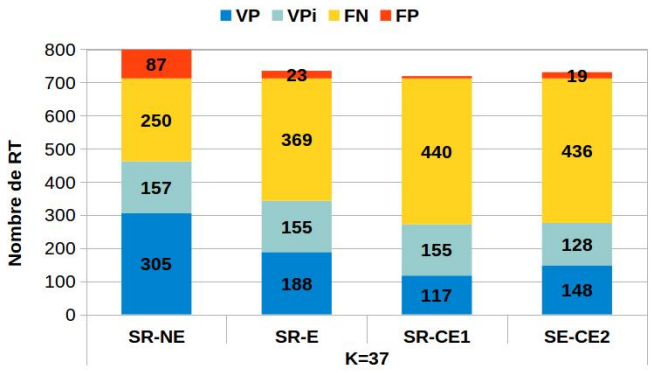
La Figure 5.7 présente les résultats obtenus sur l'ensemble des RT générales. L'efficacité d'une valeur réduite de k est une fois de plus prouvée par le pourcentage de VP obtenus. Nous pouvons perdre jusqu'à 41 % de VP quand la valeur de k varie de 17 à 47 pour un même ensemble de reads courts. Les valeurs pour la précision et la sensibilité obtenues pour chaque valeur de k sont présentées dans le Tableau 5.3. Dans ce tableau, on peut remarquer que la qualité des résultats obtenus pour $k = 17$ reste assez stable en fonction du pourcentage d'erreurs des reads courts, la valeurs de précision variant de 0,998 au 0,872. Cela est dû au fait que la forte fragmentation des reads courts permet une analyse correcte d'un nombre important de cycles, indépendamment du taux d'erreurs des reads courts.



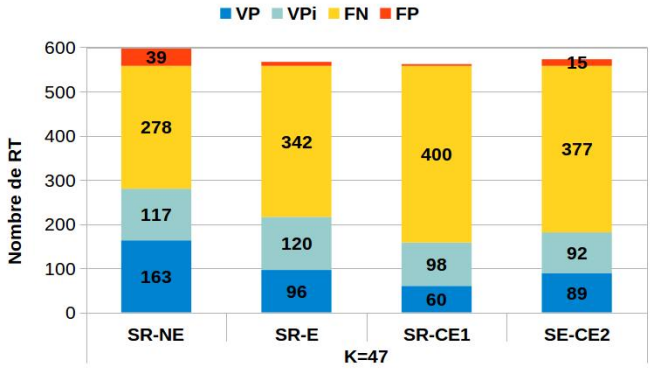
(a) Nombre de RT robustes avec une longueur de motif maximale de 100 bp obtenues pour k=17



(b) Nombre de RT robustes avec une longueur de motif maximale de 100 bp obtenues pour k=27

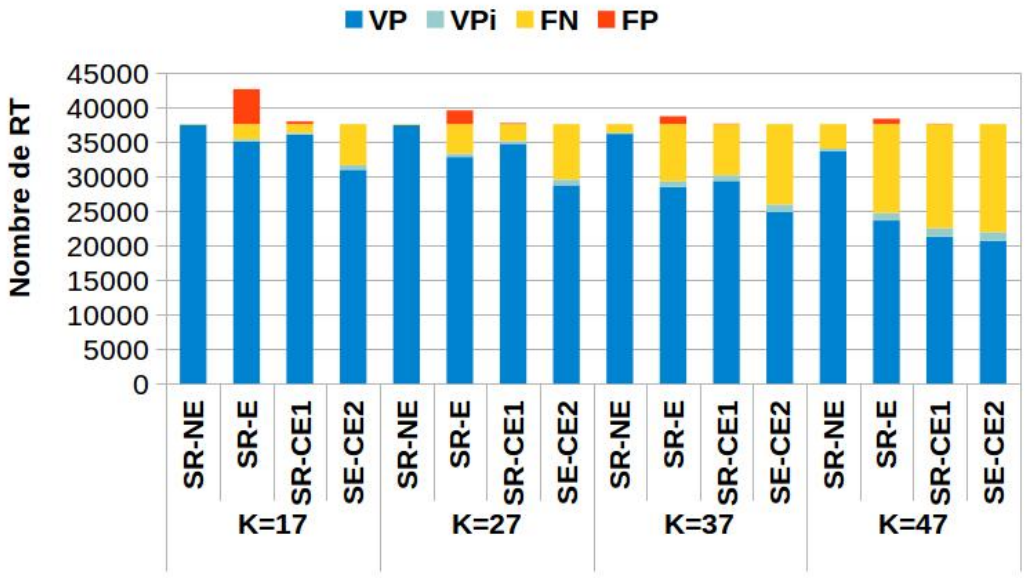


(c) Nombre de RT robustes avec une longueur de motif maximale de 100 bp obtenues pour k=37

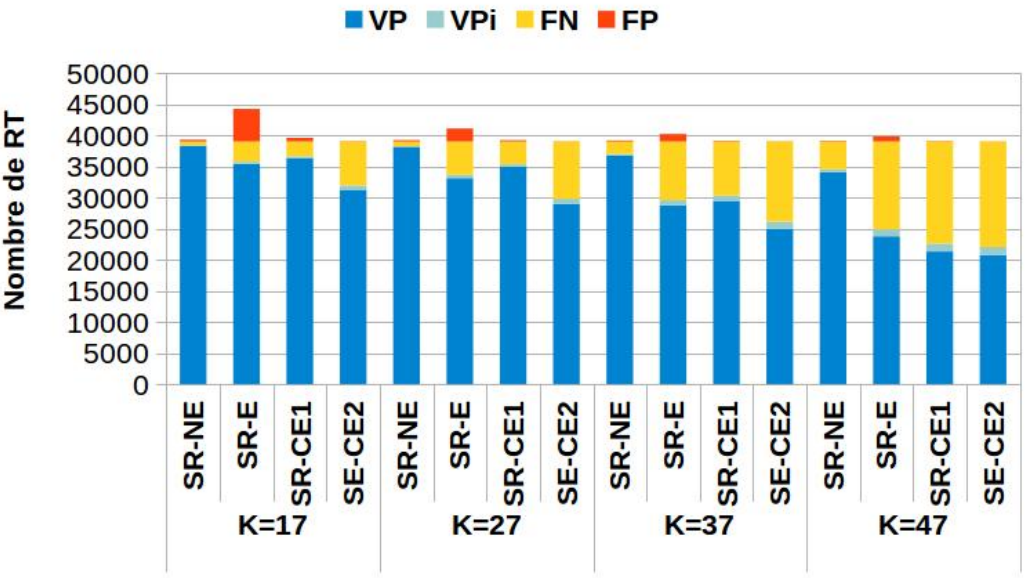


(d) Nombre de RT robustes avec une longueur de motif maximale de 100 bp obtenues pour k=47

FIGURE 5.6 – Nombre de RT robustes détectées avec des motifs ayant une longueur maximale de 100 bp pour le premier chromosome de *C. elegans*. Les résultats sont obtenus avec l'ensemble *LR-E* des reads longs avec une couverture de 20x.



(a) Nombre de RT générales avec une longueur de motif maximale de 20 bp



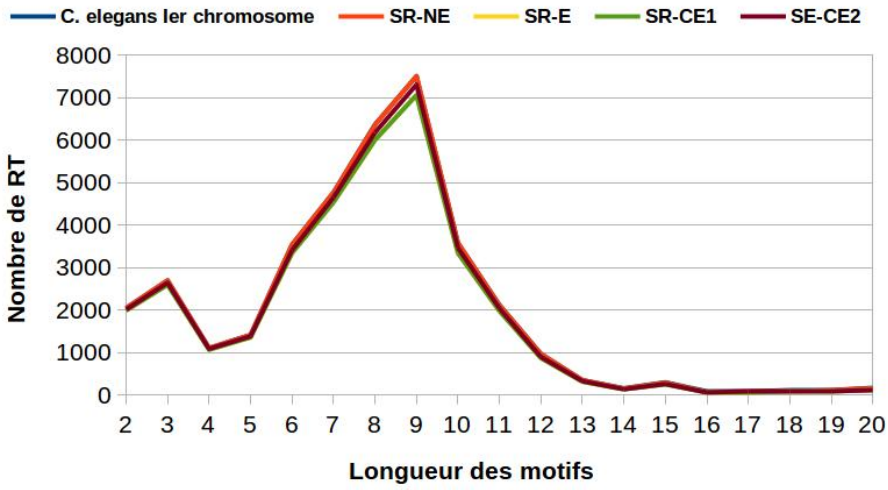
(b) Nombre de RT générales avec une longueur de motif maximale de 100 bp

FIGURE 5.7 – Nombre de RT générales détectées avec une longueur maximale de motif de 20 bp et 100 bp pour le premier chromosome de *C. elegans*. Les résultats sont obtenus avec l'ensemble *LR-E* des reads longs avec une couverture de 20x.

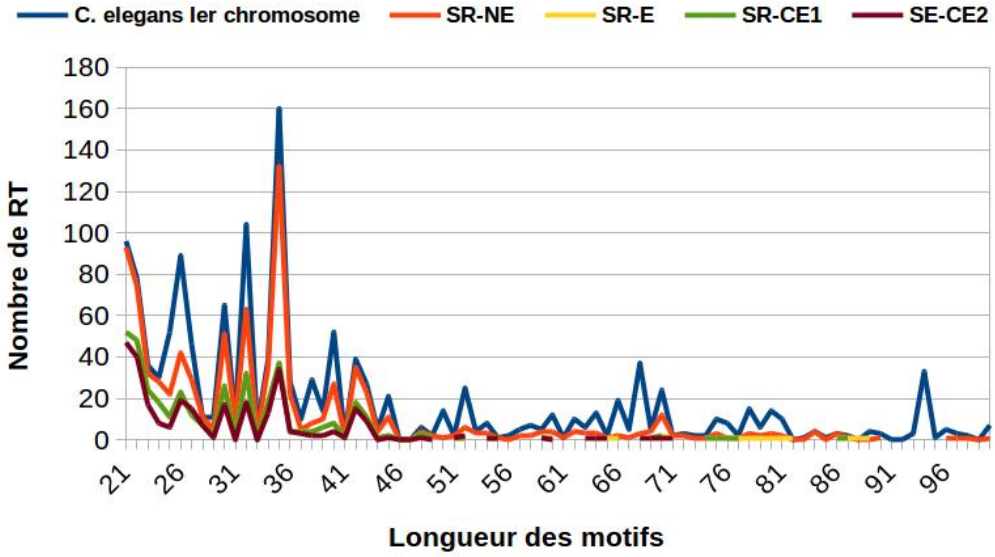
Sur la Figure 5.7 nous pouvons également observer l'effet des procédures de correction d'erreurs sur les reads courts. Le nombre de FP peut diminuer de plus de 90 % seulement en découpant les extrémités erronées des reads courts (avec l'outil de trimming de SSAKE) et même jusqu'à 100 % avec la méthode de correction d'erreurs proposée par ALLPATHS. Mais encore une fois, le nombre de VP perdus peut être important. En corrigeant les parties des reads courts avec une faible qualité, les procédures de correction d'erreurs peuvent supprimer des parties de RT correctes. L'écart des fréquences des arcs est alors augmenté dans les cycles formés par ces RT, ce qui implique des difficultés supplémentaires pour MixTaR à les identifier correctement.

Variation des longueurs des motifs des RT détectées

Dans les résultats renvoyés par mreps pour $2 \leq |p| \leq 100$, on constate que près de 96 % des RT générales du premier chromosome de *C. elegans* ont une longueur de motif respectant $2 \leq |p| \leq 20$. Mais l'intervalle $21 \leq |p| \leq 100$ est également très bien couvert, comme le chromosome a au moins une RT pour la plupart des longueurs des motifs.



(a) RT générales avec une longueur de motif maximale de 20 bp



(b) RT générales avec une longueur de motif maximale de 100 bp

FIGURE 5.8 – Distribution de la longueur des motifs des RT générales pour le premier chromosome de *C. elegans*. Les résultats obtenus avec MixTaR pour les quatre ensembles de reads courts, l'ensemble *LR-E 20x* de reads longs et k = 17 sont comparés à ceux du premier chromosome de *C. elegans*.

La Figure 5.8 décrit la répartition des RT générales en fonction de la longueur du motif pour ces deux intervalles. À côté des RT du chromosome, nous présentons les résultats obtenus avec les quatre ensembles de reads courts pour la valeur de $k = 17$ (valeur pour laquelle nous avons obtenu le plus grand nombre de RT).

Nous pouvons remarquer que sur l'intervalle de $2 \leq |p| \leq 20$, la différence sur la qualité des résultats obtenus avec les quatre ensembles différents de reads courts n'est pas

très significative. Cela est dû au fait que MixTaR détecte tous les cycles de longueur maximale de 20 nœuds pour chaque exécution. Une autre raison est la grande fragmentation des reads courts pour $k = 17$ qui permet une analyse correcte de ces cycles indépendamment du taux d'erreurs.

Néanmoins, sur l'intervalle de $21 \leq |p| \leq 100$, le nombre de RT détectées avec l'ensemble des reads courts erronés est plus réduit que celui obtenu avec l'ensemble *SR-NE* des reads courts sans erreurs. En outre, la méthode de correction d'erreurs de ALLPATHS élimine la possibilité de détecter des RT avec $|p| \geq 71$, tandis qu'avec les ensembles *SR-E* et *SR-CE1* nous détectons des RT avec une longueur de motif $|p|$ allant jusqu'à 89 bp et même jusqu'à 100 bp avec *SR-NE*. Cette limitation est due à la longueur plus courte des reads dans *SR-CE2* par rapport aux trois autres. Ainsi, en coupant les reads courts, les séquences de RT avec de longs motifs sont réduites et ne peuvent plus former des cycles dans le graphe de de Bruijn.

5.4.3 Données réelles utilisées

Pour tester la qualité des résultats de MixTaR sur des données réelles nous avons choisit la bactérie *Legionella pneumophila*, comme pour les tests réalisés avec les assembleurs de novo (Section 3.3). *Legionella pneumophila* est un parasite intracellulaire trouvé dans les monocytes chez les humains et responsable d'une pneumonie sévère connue comme la maladie du légionnaire. Plusieurs études telles que [23, 121] se sont intéressées au rôle biologique des RT du génome de *L. pneumophila*, la souche appelée *Philadelphia*.

Pour nos expériences sur la souche *Philadelphia*, nous avons utilisé le même ensemble de reads courts pairés que pour les tests effectués sur les assembleurs (Section 3.3). Pour rappel, cet ensemble de reads courts a été obtenu avec la technologie Illumina avec une couverture d'environ 190x [SRA :SRX258262]. Dans ce qui suit, nous nous référons à cet ensemble de reads courts par *SR-RP* (Short Reads Real for *Philadelphia*). Comme aucun ensemble de reads longs réels n'est disponible pour cette souche, nous avons simulé deux ensembles de reads longs avec PBSIM avec une couverture de 20x et 100x (*LR-EP* 20x et *LR-EP* 100x pour Long Reads with Errors for *Philadelphia*). Deux autres ensembles de reads longs ont été obtenus en corrigeant *LR-EP* 20x et *LR-EP* 100x avec LoRDEC utilisant l'ensemble de reads courts réels et $k = 19$. Ces ensembles sont notés *LR-CEP* 20x et *LR-CEP* 100x (Long Reads with Corrected Errors for *Philadelphia*). Le génome a une longueur d'environ 3,4 Mb [GenBank : GCA_000008485.1].

Afin de tester MixTaR avec des ensembles de reads longs et courts réels, nous avons également utilisé pour nos expériences une deuxième souche de *L. pneumophila*, la souche appelée *130b*. Pour nos expériences, nous avons utilisé deux ensembles de reads téléchargés à partir de Sequence Read Archive au NCBI : un ensemble de reads courts pairés obtenu avec le séquençage Illumina avec une couverture d'environ 120x [SRA : ERX313832] et un ensemble de reads longs obtenu avec le séquençage PacBio [SRA : ERX620205]. Dans ce qui suit, nous nous référons à ces deux ensembles de reads avec *SR-R130b* (Short Reads Real for *130b*) et *LR-R130b* (Long Reads Real for *130b*). Un deuxième ensemble de reads longs, *LR-RC130b*, (Long Reads Real Corrected for *130b*) a été obtenu en corrigeant l'ensemble *LR-R130b* avec LoRDEC utilisant l'ensemble *SR-R130b* et $k = 19$. Le génome de cette souche n'étant pas finalisé, il est constitué de 159 contigs [GenBank : GCA_000211115.2].

5.4.4 Analyse des résultats obtenus avec les données réelles

Pour les expériences que nous présentons par la suite concernant les deux souches de *L. pneumophila*, nous avons utilisé les valeurs suivantes pour les paramètres de MixTaR. Pour le paramètre k, nous avons testé toutes les valeurs impaires pour $k \in [17, 47]$, en raison de la longueur de 100 bp des reads courts. Nous présentons les résultats obtenus pour $k = 17$, la valeur pour laquelle nous avons obtenu le plus grand ensemble de RT. Étant donné que la taille du génome est nettement plus petite que la taille du premier chromosome de *C. elegans*, les tailles de l'ensemble des k-mers et du graphe de de Bruijn sont également plus petites. Par conséquent, nous avons inclus dans notre recherche les cycles dont les k-mers ont un nombre d'occurrences dans l'ensemble de reads courts de $\sigma = 30$.

Pour la recherche des cycles nous avons utilisé $\eta = 10000$ arcs pour les cycles d'une longueur maximale de $\Lambda_{max} = 100$ et $\lambda_{max} = 30$. Pour le score minimum d'alignement τ, nous avons utilisé la valeur $\tau = 10$ pour les reads longs non corrigés et $\tau = 20$ pour ceux corrigés.

Résultats obtenus pour la souche *Philadelphia* de *L. pneumophila*

Sur la souche *Philadelphia*, mreps obtient 2227 RT générales avec $2 \leq |p| \leq 100$. Parmi ces RT, seulement 17 sont robustes pour $k = 17$. En fonction de l'ensemble de reads longs utilisé, MixTaR détecte au moins 13 d'entre elles complètement, comme le montre la Figure 5.9a.

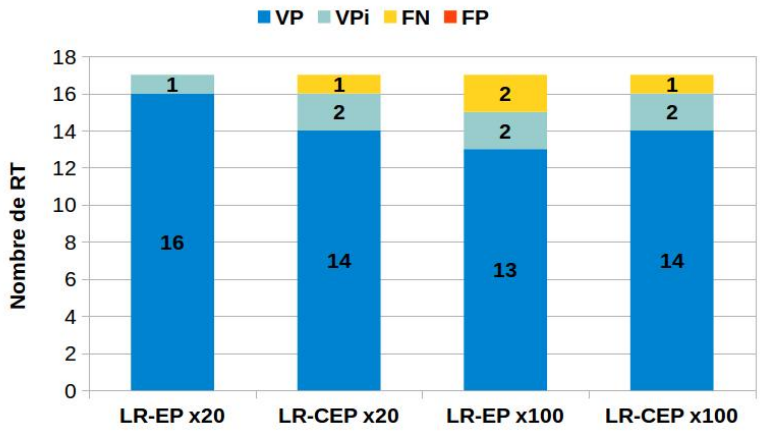
Il faut noter qu'il n'y a pas de FP. Les RT détectées par notre algorithme sont, soit complètes (VP), soit avec un nombre inférieur de copies, mais présentes sur la souche *Philadelphia* (VP_i). Cela est expliqué par la sélection des reads courts significatifs pour les assemblages locaux. Les assemblages Greedy peuvent introduire des FP lorsque deux reads chevauchants ne sont pas situés dans la même région du génome. Comme mentionné précédemment, ce cas apparaît surtout à cause des répétitions. Comme la souche *Philadelphia* contient un faible nombre de répétitions et puisque nous limitons le nombre de reads courts utilisés dans les assemblages, le nombre de FP est très faible.

Ce comportement est également remarqué pour les RT générales, avec moins de 0,75 % de FP sur les ensembles de RT générales détectées par MixTaR (voir Figure 5.9b). Toutefois, le pourcentage de RT générales détectées par notre algorithme est plus faible que celui obtenu pour *C. elegans*. Ceci s'explique par le fait que les RT générales sur la souche *Philadelphia* ne sont pas toujours situées dans les régions adjacentes de RT robustes ou n'ont pas un motif similaire à un motif d'une RT robuste. Jusqu'à 30 % des RT générales de la souche *Philadelphia* ne sont pas détectées par MixTaR.

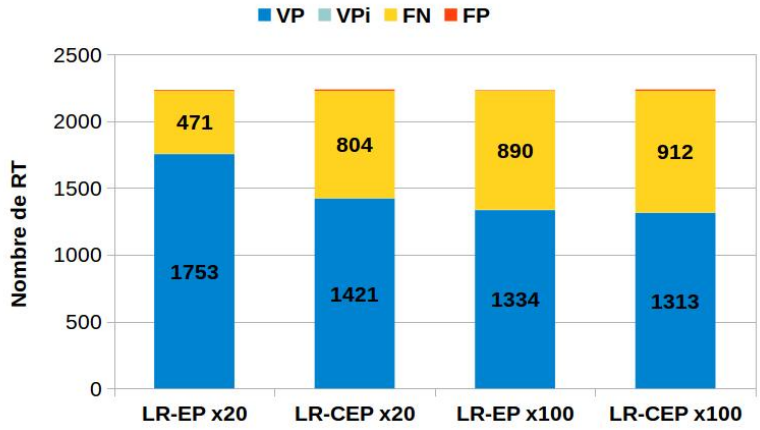
Contrairement au premier chromosome de *C. elegans*, la différence entre la qualité des résultats obtenus avec les ensembles des reads longs non corrigés et corrigés est plus importante. La différence de qualité est également soulignée par les valeurs de la sensibilité du Tableau 5.4. La raison principale est l'utilisation de reads courts réels pour la procédure de correction d'erreurs des reads longs. LoRDEC n'est pas capable de détecter toutes les erreurs dans les RT présentes dans les reads longs.

Les RT de la souche *Philadelphia* ont des motifs avec des longueurs allant de 2 à 48 bp, tel que présenté sur la Figure 5.10. En raison de la longueur des cycles analysés, notre algorithme est capable de couvrir la totalité de l'intervalle de longueur des motifs.

Plusieurs RT de la souche *Philadelphia* de *L. pneumophila* ont été étudiées pour leur signification biologique [23, 121]. Nous présentons dans le Tableau 5.5 les résultats obte-



(a) Nombre de RT robustes



(b) Nombre de RT générales

FIGURE 5.9 – Nombres de RT robustes et de RT générales détectées pour le génome de *L. pneumophila* (souche *Philadelphia*). Les résultats sont obtenus avec l'ensemble *SR-RP* des reads courts réels, k = 17 et les différents ensembles de reads longs.

TABLEAU 5.4 – Précision et sensibilité de la détection des RT robustes et des RT générales pour le génome de *L. pneumophila* (souche *Philadelphia*). En gras, les meilleures valeurs obtenues pour chaque catégorie de RT.

	RT robustes		**RT générales**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
LR-EP 20x	**1**	**1**	**0.997**	**0.789**
LR-CEP 20x	**1**	0.941	0.993	0.639
LR-EP 100x	**1**	0.882	**0.997**	0.600
LR-CEP 100x	**1**	0.941	0.992	0.590

nus par MixTaR pour toutes les RT significatives avec $2 \leq |p| \leq 100$ qui sont présentées dans [23, 121].

L'ensemble des RT utilisées dans les résultats présentés auparavant a été obtenu par mreps avec la plus grande valeur pour le score d'alignement entre les copies d'une RTA. Les articles [23, 121] ne précisent pas les séquences exactes des RT analysées, mais

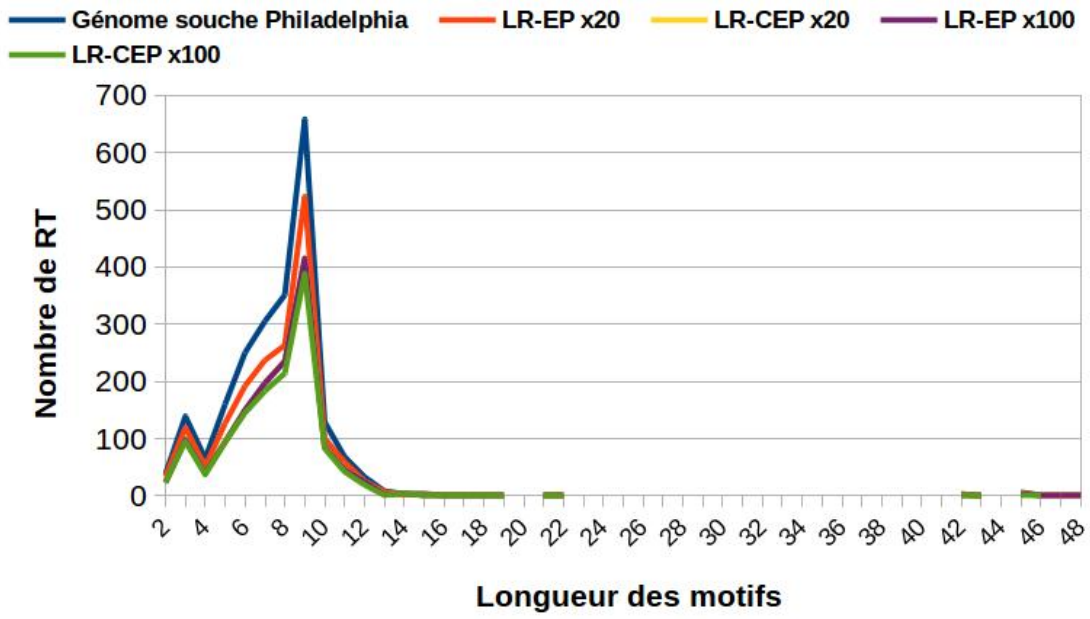


FIGURE 5.10 – Distribution de la longueur des motifs des RT générales détectées pour le génome de *L. pneumophila* (souche *Philadelphia*). Les résultats obtenus avec MixTaR avec les quatre ensembles de reads longs, l'ensemble *SR-RP* des reads courts réels et k= 17 sont comparés à l'ensemble des RT de la souche *Philadelphia*.

seulement l'emplacement approximatif sur la souche (le gène), la longueur du motif et le nombre de copies. Ainsi, afin de récupérer leur séquence, nous exécutons mreps sur les régions déterminées (gènes) de la souche *Philadelphia* et nous avons diminué le paramètre pour le score d'alignement jusqu'à trouver une RT correspondant à la longueur du motif et au nombre de copies fournies dans [23, 121].

Le nom des RT et des gènes présentés dans le Tableau 5.5 sont ceux mentionnés dans [23, 121]. Nous avons ensuite cherché leurs séquences dans les résultats obtenus par MixTaR, et, tel que présenté dans le Tableau 5.5, la plupart des RT sont complètement détectées par notre algorithme, notamment avec l'ensemble de reads longs *LR-CEP 20x*.

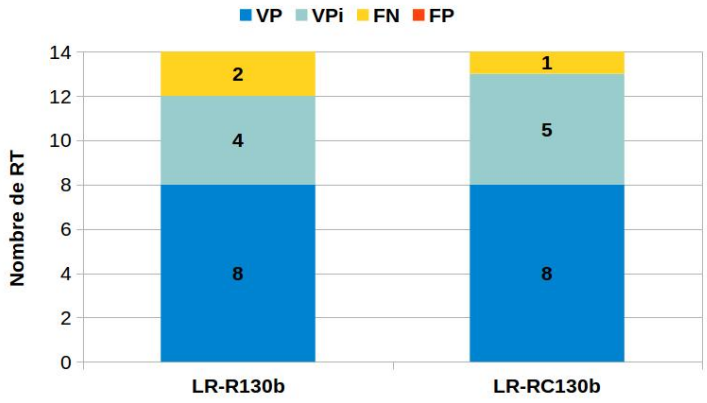
Résultats obtenus pour la souche *130b* de *L. pneumophila*

Après l'exécution de mreps sur les contigs de la souche *130b* pour détecter les RT avec $2 \leq |p| \leq 100$, nous avons obtenu 2230 RT générales. Seulement 14 d'entre elles sont des RT robustes pour $k = 17$. Comme pour la souche *Philadelphia*, MixTaR obtient des résultats satisfaisants (voir la Figure 5.11). Une fois de plus, l'algorithme ne renvoie pas de FP, et les RT détectées sont, soit complètes (VP), soit avec un nombre inférieur de copies, mais toutefois présentes sur la souche *130b* (VP_i).

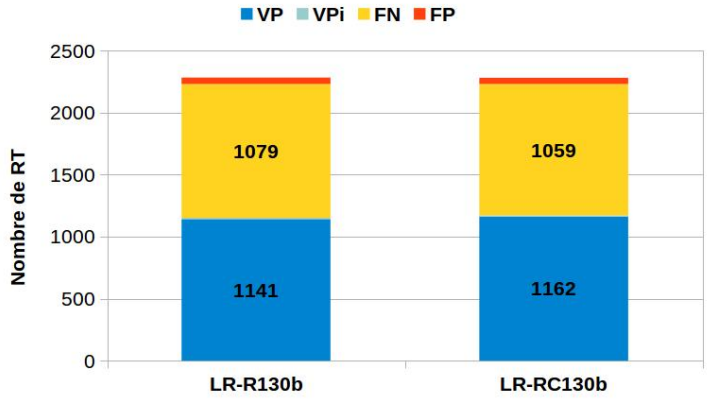
En dépit du petit nombre de RT qui peuvent former un cycle dans le graphe de de Bruijn pour $k = 17$, MixTaR est capable de détecter plus de 51 % de toutes les RT générales détectées par mreps sur le génome de la souche *130b* (voir la Figure 5.11). Comme auparavant, les RT générales qui ne sont pas retrouvées par MixTaR ont des motifs très différents de ceux des RT robustes ou sont situées à une distance importante de celles-ci. De plus, la diminution du nombre de RT robustes de la souche *130b* par rapport à la souche *Philadelphia* réduit les chances pour MixTaR de détecter les RT générales.

TABLEAU 5.5 – Résultats de MixTaR concernant la résolution des RT biologiquement significatives pour le génome de *L. pneumophila* (souche *Philadelphia*) présentées dans [23, 121].

Nom	**Gène**	**Longueur du motif**	**Nombre de copies**	**LR-EP 20x**	**LR-CEP 20x**	**LR-EP 100x**	**LR-CEP 100x**
	LPG0451	30 bp	5.9	Complete	-	-	-
	LPG0688	9 bp	6	Complete	Complete	Complete	Complete
	LPG1038	12 bp	4.17	Complete	Complete	Complete	Complete
Lpms35	LPG1299	18 bp	3	Complete	-	-	-
	LPG1555	21 bp	2	Complete	Complete	Complete	Complete
	LPG1602	90 bp	9.2	Complete	Complete	Complete	Complete
	LPG1948	90 bp	7.08	Complete	-	Complete	-
	LPG1958	87 bp	13.59	Complete	-	Complete	-
	LPG2392	87 bp	6.49	-	-	-	-
	LPG2559	12 bp	4.08	Complete	Complete	Complete	Complete
Lpms31	LPG2644	45 bp	19.44	Incomplete (11 copies)	Incomplete (11 copies)	Incomplete (11 copies)	Incomplete (11 copies)
Lpms3	LPG2793	96 bp	7.58	Complete	-	-	-
Lpms01	LPG2854	45 bp	7.64	Incomplete (6 copies)	-	-	-
Lpms13	LPG1488	24 bp	9.75	Incomplete (6 copies)	Incomplete (6 copies)	Incomplete (6 copies)	Incomplete (6 copies)
Lpms17	LPG0854	89 bp	2.28	-	Complete	-	Complete
Lpms19	Intergenic	21 bp	4.05	-	Complete	-	Complete



(a) Nombre de RT robustes



(b) Nombre de RT générales

FIGURE 5.11 – Nombres de RT robustes et de RT générales détectées pour le génome de *L. pneumophila* (souche *130b*). Les résultats sont obtenus avec l'ensemble *SR-R130b* de reads courts réels et $k = 17$.

TABLEAU 5.6 – Précision et sensibilité de la détection des RT robustes et des RT générales pour le génome de *L. pneumophila* (souche *130b*). En gras, les meilleures valeurs obtenues pour chaque catégorie de RT.

	RT robustes		**RT générales**	
	Précision	**Sensibilité**	**Précision**	**Sensibilité**
LR-R130b	**1**	0.857	0.957	0.516
LR-RC130b	**1**	**0.929**	**0.959**	**0.525**

Les résultats obtenus par notre algorithme contiennent un faible pourcentage de FP. Avec l'ensemble *LR-R130b* seulement 8,8 % des RT détectées sont des FP et la valeur diminue à 8,6 % pour l'ensemble *LR-RC130b*.

Les valeurs de la précision et de la sensibilité pour les deux ensembles de RT sont présentées dans le Tableau 5.6. Contrairement aux résultats obtenus avec les reads longs sur la souche *Philadelphia*, MixTaR obtient des résultats légèrement meilleurs avecs l'ensemble des reads longs corrigés qu'avec l'ensemble des reads longs non corrigés. Ceci est probablement dû à un taux d'erreurs des reads longs réels supérieur à celui des reads

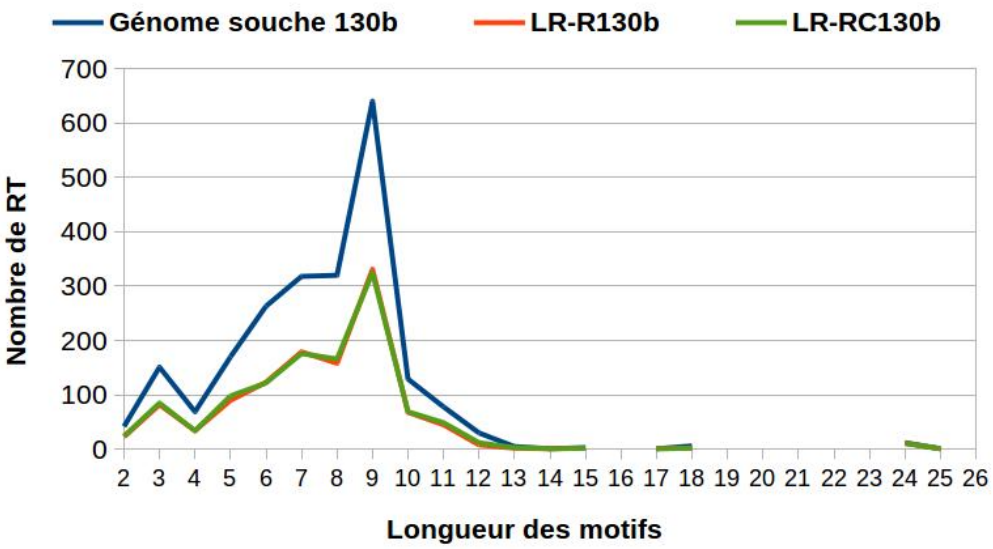


FIGURE 5.12 – Distribution de la longueur des motifs des RT générales détectées pour le génome de *L. pneumophila* (souche *130b*). Les résultats obtenus avec MixTaR avec les deux ensembles de reads longs, l'ensemble *SR-R130b* des reads courts réels et $k = 17$ sont comparés à l'ensemble des RT de la souche *130b*.

longs simulés.

Les RT de *L. pneumophila* ont des motifs de longueur allant de 2 à 25 bp, tel que présenté sur la Figure 5.12. Encore une fois, en raison de la longueur des cycles analysés, les résultats obtenus avec les deux ensembles de reads longs couvrent la totalité de l'intervalle de longueur des motifs.

5.5 Conclusion

Ce chapitre présente l'algorithme MixTaR qui représente une solution efficace au problème de la détection de novo de RT. Le procédé ne se concentre que sur les parties du génome où des RT potentielles peuvent se trouver, et ne calcule pas un assemblage global. En tirant parti de la qualité des reads courts et de la longueur des reads longs, MixTaR est une approche fiable. Nous avons obtenu des résultats de haute qualité sur les organismes complexes, en utilisant des ensembles de reads simulés et réels, avec différents taux d'erreurs. En conservant de faibles taux de faux positifs, notre méthode détecte des RT avec des longueurs de motifs variant dans un large intervalle.

MixTaR identifie un nombre important de RT avec les deux types d'ensembles de reads réels et simulés. Toutefois, la complexité du fragment d'ADN cible influence la quantité de RT détectées. Comme mentionné précédemment, l'objectif initial de notre approche était l'identification des RT robustes. En assemblant les reads courts chevauchant les motifs de RT robustes, nous sommes en mesure d'identifier une quantité importante de RT non robustes. Cela est dû soit à la position de ces dernières dans les régions voisines des RT robustes, soit à la similarité entre les motifs des RT non robustes et ceux des RT robustes. En conséquence, le nombre de RT générales détectées croît avec le nombre de RT robustes présentes dans le fragment d'ADN cible. Ainsi, les performances de MixTaR augmentent avec la complexité du fragment d'ADN cible.

Des améliorations et des extensions futures sont prévues pour notre algorithme et pour notre étude. Chaque outil qui détecte les RT dans des fragments d'ADN connus a sa propre définition pour les RTA, ainsi les ensembles de RT détectées sont différents [71]. Pour nos tests, nous avons utilisé *mreps* [60], mais l'étude peut être étendue à d'autres outils de recherche de RT, comme par exemple *Tandem repeat finder* [7].

En outre, dans ce chapitre, les résultats sont présentés du point de vue des séquences de RT. L'algorithme ainsi que l'étude de la qualité des RT détectées peuvent être étendus pour inclure les régions adjacentes des RT. De ce fait, au lieu de seulement comparer les séquences des RT entre nos résultats et la séquence d'ADN cible, nous pouvons également fournir plus d'informations concernant l'emplacement de chaque RT.

Une autre amélioration envisagée est la généralisation des RT robustes. Pour MixTaR, une RT robuste contient une RTE interne formant un cycle dans le graphe de de Bruijn. Mais comme présenté pour DExTaR, toute répétition d'un motif ayant une longueur minimale de k forme un cycle dans ce graphe. Ainsi, nous envisageons d'étendre la définition des RT robustes aux RT contenant une répétition exacte qui n'est pas nécessairement en tandem, avec un motif de longueur minimale k.

III

Problème d'assemblage avec des reads pairés : le graphe de de Bruijn pairé

6

Algorithmes d'assemblage avec le graphe de de Bruijn pairé

Dans le but d'améliorer l'assemblage des répétitions nous nous sommes intéressés également au problème d'assemblage basé sur le *graphe de de Bruijn pairé*. Ce nouveau graphe a été proposé dans la recherche de méthodes plus efficaces pour un assemblage correct et complet avec des reads courts [82]. Comme précisé auparavant, de nombreuses technologies de séquençage de nouvelle génération sont capables de générer des *reads pairés*, c'est-à-dire des paires de reads séparés par une distance connue nommée *taille d'insert*. La taille d'insert étant beaucoup plus grande que la longueur des reads courts, les reads pairés peuvent être utilisés pour connecter des régions contenant de longues répétitions.

Les assembleurs de de Bruijn classiques utilisent les informations apportées par les reads courts pairés uniquement lors des étapes de post-traitement, comme par exemple à l'étape de scaffolding. Le graphe de de Bruijn pairé intègre ces informations directement dans sa structure [82]. Il est basé sur le graphe de de Bruijn classique, mais les chevauchements sont calculés entre des paires de k-mers séparés par la taille d'insert des reads pairés. Cela améliore la qualité de l'assemblage et facilite la détection de répétitions [82]. En contrepartie, les problèmes deviennent algorithmiquement beaucoup plus difficiles à traiter que sur les graphes de de Bruijn classiques. En effet, le PROBLÈME DU CYCLE COUVRANT SOLIDE que nous définissons ci-dessous est NP-difficile [57].

Ce chapitre introduit des notions propres au *graphe de de Bruijn pairé* et décrit nos travaux axés sur l'assemblage avec ce type de graphe. Nous définissons tout d'abord sa structure et le problème d'assemblage avec ce graphe. Ensuite, nous présentons les algorithmes exacts proposés pour le PROBLÈME DU CYCLE COUVRANT SOLIDE et pour plusieurs variantes de ce problème.

Ces travaux ont été réalisés en collaboration avec Christian Komusiewicz et ont été publiés dans [63].

6.1 Définitions

Dans cette section, nous définissons les notations utilisées ainsi que la façon dont le graphe de de Bruijn pairé est construit à partir des données d'entrée.

On considère le graphe orienté $G = (V, A)$ pour lequel V désigne l'ensemble des nœuds et A l'ensemble des arcs. Un *chemin* $(v_1, \ldots, v_q)$ est un n-uplet de nœuds tel que $(v_i, v_{i+1}) \in A$, $1 \leq i < q$. La *longueur* $|W|$ d'un chemin $W = (v_1, \ldots, v_q)$ est le nombre q des éléments dans le n-uplet. Un chemin est *simple* si $i \neq j$ implique $v_i \neq v_j$. Pour un chemin $W = (v_1, \ldots, v_q)$, le i-ème nœud de W est noté $W[i] = v_i$. Nous définissons, un *cycle* dans un graphe $G = (V, A)$ comme un chemin $(v_1, \ldots, v_q)$, $v_i \in V$, tel que $(v_q, v_1) \in A$. Enfin, l'ensemble des arcs du chemin W est noté $A(W)$.

Étant donnés deux chemins $W_1 = (v_1, \ldots, v_q)$ et $W_2 = (u_1, \ldots, u_t)$ tels que $(v_q, u_1) \in A$, la *concaténation* de W_1 et W_2 est notée $W_1 \cdot W_2 = (v_1, \ldots, v_q, u_1, \ldots, u_t)$. Soit un cycle W et un entier n, la notation W^n représente le cycle obtenu en concaténant n fois W avec lui-même.

Pour la construction d'un graphe de de Bruijn pairé, nous avons besoin d'un ensemble $R = \{(r_1^{\mathcal{L}}, r_1^{\mathcal{R}}), \ldots, (r_m^{\mathcal{L}}, r_m^{\mathcal{R}})\}$ de reads pairés et de deux entiers d et k. Chaque $(r_i^{\mathcal{L}}, r_i^{\mathcal{R}})$ est une paire de séquences ayant la même longueur sur l'alphabet $\Sigma = \{A, C, G, T\}$. L'entier d représente la *taille d'insert*, également appelée *décalage*. Il précise que les deux reads d'une paire sont deux sous-séquences du fragment d'ADN cible séparées par une distance de d bp. Cela veut dire que $r_i^{\mathcal{L}} + s + r_i^{\mathcal{R}}$ est une sous-séquence du fragment d'ADN cible où $|s| = d$ pour toute paire $(r_i^{\mathcal{L}}, r_i^{\mathcal{R}}) \in R$. L'entier k représente, comme pour le graphe de de Bruijn classique, la longueur des k-mers.

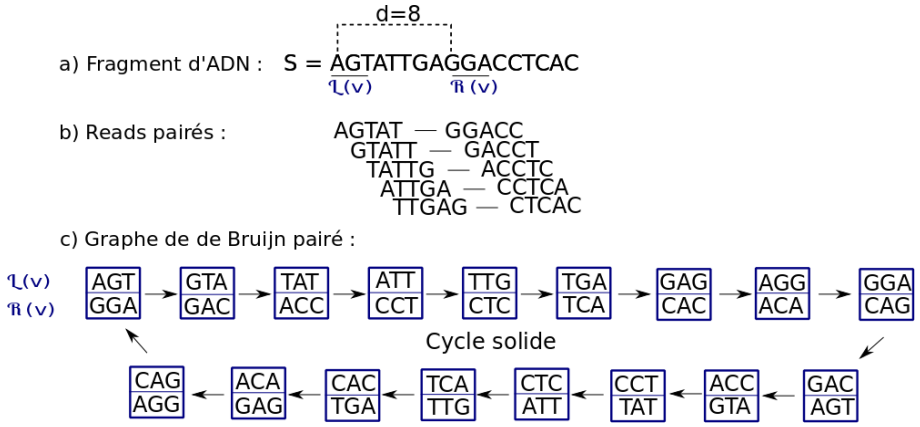


FIGURE 6.1 – Exemple de graphe de de Bruijn pairé avec un cycle *solide* : a) Le fragment d'ADN cible. b) L'ensemble des reads pairés avec la taille d'insert d=8. c) Le graphe de de Bruijn pairé construit à partir des reads pairés du b) pour k=3. La partie supérieure de chaque nœud v est représentée par l'étiquette $\mathcal{L}(v)$, la partie inférieure est représentée par l'étiquette $\mathcal{R}(v)$.

Dans un graphe de de Bruijn pairé $G^k(R)$ construit à partir d'un ensemble de reads pairés R, chaque nœud v est associé à une paire de k-mers $(\mathcal{L}(v), \mathcal{R}(v))$ (voir Figure 6.1 pour un exemple). Cette paire est appelée *bilabel* et chaque nœud possède une bilabel unique. Pour un ensemble de reads pairés R, l'ensemble des nœuds de $G^k(R)$ est défini de la manière suivante :

$$V(R) = \{(s,t) \in \Sigma^k \times \Sigma^k \mid \exists (r_i^{\mathcal{L}}, r_i^{\mathcal{R}}) \in R, q \in \mathbb{N} : s = r_i^{\mathcal{L}}[q,k] \wedge t = r_i^{\mathcal{R}}[q,k]\}. \quad (6.1)$$

Un arc est construit à partir d'un nœud u vers un nœud v si deux reads dans R contiennent les bilabels de u et v à des positions consécutives : un read r_1 pour $\mathcal{L}(u)$ et $\mathcal{L}(v)$ et un autre read r_2, situé à une distance d de r_1, pour $\mathcal{R}(u)$ et $\mathcal{R}(v)$. Plus précisément, l'ensemble des arcs de $G^k(R)$ est défini de la manière suivante :

$$\begin{aligned} A(R) = \{(u,v) \mid & \exists (r_i^{\mathcal{L}}, r_i^{\mathcal{R}}) \in R, q \in \mathbb{N} : \\ & \mathcal{L}(u) = r_i^{\mathcal{L}}[q,k] \wedge \mathcal{L}(v) = r_i^{\mathcal{L}}[q+1,k] \wedge \\ & \mathcal{R}(u) = r_i^{\mathcal{R}}[q,k] \wedge \mathcal{R}(v) = r_i^{\mathcal{R}}[q+1,k]\}. \end{aligned}$$

Contrairement aux graphes de de Bruijn classiques, un chemin dans un graphe de de Bruijn pairé correspond à deux séquences sur l'alphabet Σ et donc à deux séquences d'ADN. Plus précisément, un chemin $W = (v_1, \ldots, v_q)$ dans un graphe de de Bruijn pairé correspond à deux séquences $\mathcal{L}(v_1) \cdot \mathcal{L}(v_2)[k] \cdot \ldots \cdot \mathcal{L}(v_q)[k]$ et $\mathcal{R}(v_1) \cdot \mathcal{R}(v_2)[k] \cdot \ldots \cdot \mathcal{R}(v_q)[k]$. Soit D le fragment d'ADN cible qui a été séquencé pour obtenir l'ensemble R des reads pairés, alors il existe un seul chemin dans $G^k(R)$ qui correspond au fragment D. Ce chemin satisfait les propriétés décrites par la suite. L'objectif que nous considérons ici est de décider si un chemin remplissant ces propriétés existe dans $G^k(R)$.

6.2 Le problème du cycle couvrant solide

Dans le cas d'organismes avec un seul chromosome circulaire, le chemin correspondant au génome est un cycle. Ce cycle doit avoir la longueur estimée du génome notée ℓ. Comme présenté précédemment, le cycle correspond à deux séquences cycliques. Le cycle qui correspond au génome doit contenir chaque paire de $k+1$-mers. La construction de $G^k(R)$ fait qu'une paire de $k+1$-mers dans l'ensemble des reads correspond à un arc entre deux nœuds. Ainsi, nous exigeons que le cycle contienne tous les arcs du graphe. Par conséquent, un cycle $c = (v_1, \ldots, v_q)$ est appelé *couvrant* si, pour chaque arc $(u,w) \in A$, il existe un nœud v_i du cycle c tel que $(u,w) = (v_i, v_{i+1})$ ou $(u,w) = (v_q, v_1)$. Les propriétés énoncées ci-dessus sont également valables dans les graphes de de Bruijn classiques. Par la suite, nous présentons les propriétés spécifiques au graphe de de Bruijn pairé.

Dans un graphe de de Bruijn pairé, les chemins doivent respecter la contrainte liée à la taille d'insert. Rappelons que la distance entre les étiquettes $\mathcal{L}(v)$ et $\mathcal{R}(v)$ d'un nœud v est d. La contrainte de solidité, définie par la suite, garantit que les séquences correspondant à ces deux étiquettes sont compatibles avec la taille d'insert. Plus précisément, dans un graphe de de Bruijn pairé, un cycle est appelé *solide* si les deux séquences lui correspondant ont un décalage de d caractères. Soient s et t ces deux séquences correspondant au cycle c. Si $d \leq \ell$, alors le cycle c est *solide* si :

- $s[i+d] = t[i]$ pour $1 \leq i \leq \ell - d$, et
- $s[i] = t[i+\ell-d]$ pour $1 \leq i \leq d$.

Également, un nœud v_i dans un chemin $(v_1, \ldots, v_q)$ est considéré comme *solide* si $\mathcal{R}(v_i) = \mathcal{L}(v_{i+d})$. La définition de la solidité correspond à la définition présentée dans [57] et ne couvre que le cas où $d \leq \ell$. Cependant dans les applications, la longueur d'un cycle solide peut être inférieure au décalage d, c'est-à-dire $d > \ell$. Soient s et t les deux séquences cycliques correspondant à un cycle de longueur ℓ. Comme $d > \ell$, on considère $i + d > \ell$. La séquence s étant cyclique, on déduit que $s[i + d] = s[i + (d \mod \ell)]$. Ainsi, pour qu'un cycle soit solide nous exigeons maintenant que ses deux séquences s et t respectent les conditions suivantes :

- $s[i + (d \mod \ell)] = t[i]$ pour $1 \leq i \leq \ell - (d \mod \ell)$, et
- $s[i] = t[i + \ell - (d \mod \ell)]$ pour $1 \leq i \leq (d \mod \ell)$.

Ceci est équivalent à la condition que le cycle soit solide pour un décalage $d \mod \ell$. Par conséquent, nous supposons que $d < \ell$ par la suite, car tous les cas pour lequels $d > \ell$ peuvent être transformés en temps linéaire en une condition équivalente avec $d \leq \ell$.

Cela conduit à notre définition du problème principal :

> PROBLÈME DU CYCLE COUVRANT SOLIDE
> **Entrée :** Les entiers positifs k, d et ℓ et le graphe de de Bruijn pairé $G^k(R)$ correspondant à l'ensemble R de reads pairés avec une taille d'insert d et provenant d'un fragment d'ADN cible de longueur estimée ℓ.
> **Question :** Le graphe $G^k(R)$ contient-il un cycle couvrant solide avec un décalage de d caractères et de longueur ℓ ?

L'article [57] démontre que le PROBLÈME DU CYCLE COUVRANT SOLIDE est NP-dur même dans le cas où les variables d ou k sont des constantes ayant des valeurs très réduites. Les auteurs affirment également que, pour des valeurs constantes de $|\Sigma|$ et de k, le graphe possède une taille constante et donc le PROBLÈME DU CYCLE COUVRANT SOLIDE ne peut pas être NP-dur comme le langage définit est sparse dans le cas où d est encodé en unaire.

Nous ne faisons pas cette hypothèse et donc, dans notre cas, d et ℓ sont codés en binaire. Ainsi, la complexité du PROBLÈME DU CYCLE COUVRANT SOLIDE pour des graphes de taille fixe reste ouvert dans notre encodage. Un problème classique associé à notre PROBLÈME DU CYCLE COUVRANT SOLIDE est le calcul du cycle couvrant de longueur maximum ℓ dans un graphe orienté. Ce problème est connu sous le nom du PROBLÈME DU POSTIER CHINOIS et peut être résolu dans un temps polynomial dans le cas d'un graphe orienté [31]. Par la suite, nous présentons nos solutions proposées pour PROBLÈME DU CYCLE COUVRANT SOLIDE basées sur les principes des résolutions du PROBLÈME DU POSTIER CHINOIS.

Tout d'abord, nous décrivons une décomposition des cycles dans les graphes orientés et un algorithme pour calculer un cycle couvrant de longueur fixe que nous utilisons tout au long de ce travail. Ensuite, nous présentons des algorithmes pour le PROBLÈME DU CYCLE COUVRANT SOLIDE pour deux cas : le cas théorique où la taille d'insert est identique pour chaque paire de reads et le cas pratique où la taille d'insert est variable. Enfin, nous décrivons un algorithme paramétré pour le problème d'assemblage dans un cas particulier de graphe de de Bruijn pairé tout en proposant une nouvelle approche pour la modélisation des graphes de de Bruijn pairés.

Dans la suite de ce chapitre, l'ensemble de reads pairés R n'est pas utile pour les problèmes de calcul. Ainsi, on note le graphe de de Bruijn pairé $G = (V, A)$ à la place de $G^k(R)$.

6.3 Décompositions en cycles/chemins

Pour décrire les algorithmes proposés pour le PROBLÈME DU CYCLE COUVRANT SOLIDE, nous avons besoin tout d'abord d'estimer la complexité pour énumérer les chemins de longueur fixe. L'algorithme derrière cette estimation est une procédure standard d'énumération.

Lemme 1. *Soit $G = (V, A)$ un graphe orienté ayant n nœuds avec le degré sortant maximum Δ et soit ℓ un entier. Il existe au plus $n \cdot \Delta^{\ell-1}$ chemins et cycles de longueur ℓ dans G et ils peuvent être énumérés en $O(n \cdot \Delta^{\ell-1} \cdot (\ell + \Delta))$.*

Démonstration. L'algorithme est le suivant. Pour chaque nœud v dans G, on énumère tous les chemins qui commencent avec v et qui ont une longueur maximale de ℓ. Cela revient à énumérer récursivement tous les chemins qui commencent avec v de la façon suivante : pour chaque chemin courant $\pi = (v_1, \ldots, v_i)$, vérifier si $|\pi| < \ell$. Si oui, alors pour chaque arc (v_i, u) dans G, il faut énumérer tous les chemins permettant de lire $(v_1, \ldots, v_i, u)$. Sinon, $|\pi| = \ell$. Dans ce cas, on renvoie π si le but est de retrouver les chemins. Si l'objectif est de renvoyer les cycles, alors on vérifie si π est un cycle, c'est-à-dire, si $(v_i, v_1) \in A$. Si oui, alors on renvoie π. Ceci conclut la description de l'algorithme. Il nous reste à trouver sa complexité maximale et le nombre d'objets énumérés.

Comme dans le graphe $G = (V, A)$, $|V| = n$, il existe n préfixes initiaux de longueur 1. Chaque chemin initial de longueur 1 peut être prolongé $\ell - 1$ fois et chaque nœud a au plus Δ arcs sortants. Ainsi, le nombre de feuilles dans l'arbre de recherche décrite ci-dessus est d'au plus $n \cdot \Delta^{\ell-1}$ et la taille de l'arbre de recherche est de $O(n \cdot \Delta^{\ell-1})$. À chaque nœud interne, le temps d'exécution est de $O(\Delta)$. Dans les feuilles de l'arbre de recherche, on a besoin de retourner le préfixe qui peut être fait en $O(\ell)$, et dans le cas des cycles, on doit vérifier si $(v_i, v_1) \in A$ ce qui peut être fait en $O(\Delta)$. Le temps global d'exécution est donc de $O(n \cdot \Delta^{\ell-1} \cdot (\ell + \Delta))$. Le nombre d'éléments énumérés est au plus le nombre de feuilles dans l'arbre de recherche. □

Ce lemme implique le lemme suivant sur le nombre de chemins simples.

Lemme 2. *Un graphe orienté $G = (V, A)$ avec n nœuds ayant le degré sortant maximum $\Delta \geq 2$ a au plus $2n \cdot \Delta^{n-1}$ chemins simples différents.*

Démonstration. Comme G a n nœuds, chaque chemin simple possède une longueur maximum de n. En utilisant la complexité fournie par le Lemme 1, le nombre de chemins simples est d'au plus $\sum_{\ell=1}^{n} n \cdot \Delta^{\ell-1} = n \cdot \sum_{\ell=1}^{n} \Delta^{\ell-1}$.

Soit $S = \sum_{\ell=1}^{n} \Delta^{\ell-1}$, alors $S - \Delta S = (1 - \Delta^n)$ et donc $S = (\Delta^n - 1)/(\Delta - 1) \leq \Delta^n/(\Delta - 1)$. Comme $\Delta \geq 2$, on déduit $\Delta/2 \leq \Delta - 1$.

Par conséquent, $n \cdot \sum_{\ell=1}^{n} \Delta^{\ell-1} \leq n \cdot \Delta^n/(\Delta - 1) \leq n \cdot \Delta^n/(\Delta/2) = 2n \cdot \Delta^{n-1}$. □

Avant de présenter nos algorithmes, nous décrivons une représentation structurée des cycles et des chemins. Tout d'abord, nous montrons que nous pouvons décomposer tout chemin ou cycle C d'un graphe G en chemins simples maximaux (notés avec Ω_i) liés par des chemins, éventuellement vides (notés avec W_i). Ici, le terme maximal fait référence à la propriété que chaque chemin $\Omega_i = (u_1, \ldots, u_t)$ dans C est suivi par son premier nœud u_1. Cela implique en particulier que Ω_i est un cycle selon notre définition. Puis, nous montrons qu'il existe des décompositions avec un nombre limité de chemins/cycles et que la longueur des cycles dans ces décompositions suffit à déterminer la longueur totale du cycle décomposé. Cette propriété est ensuite utilisée afin de limiter le temps d'exécution pour déterminer l'existence d'un cycle couvrant de longueur ℓ dans G.

Lemme 3. *Soit C un cycle dans un graphe G. Alors C peut être écrit comme une concaténation de cycles Ω_i et de chemins simples W_i tels que $C = \Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ et*

1. *$|\Omega_i| > 0$ pour chaque $i \in \{1, \ldots, q\}$, et*
2. *pour chaque $\Omega_i = (u_1, u_2, \ldots, u_s)$, $1 \leq i < q$ cela signifie que $u_1 = v_1$ où v_1 est le premier nœud de $W_i \cdot \Omega_{i+1}$.*

Démonstration. Soit $C = (c_1, c_2, \ldots, c_p)$ un cycle. Si C est un cycle simple, alors en définissant $C = \Omega_1$, on obtient la décomposition désirée. Sinon, nous construisons la représentation recherchée de C comme suit.

Étant donné que C n'est pas un cycle simple, il existe au moins un nœud qui apparaît deux fois dans C. Ainsi, on suppose, sans perte de généralité, que $(c_1, \ldots, c_r)$ est un cycle simple et que $c_{r+1} = c_1$. Alors, on définit $\Omega_1 = (c_1, \ldots, c_r)$. Si $(c_{r+1}, \ldots, c_p)$ est un cycle simple, alors, en fixant $(c_{r+1}, \ldots, c_p) = \Omega_2$, on obtient la décomposition recherchée (car $c_1 = c_{r+1}$). Sinon, il existe une paire de nœuds c_s et c_t telle que $c_s = c_t$, $s < t$, et t est le minimum parmi tous les indices remplissant cette condition. Alors, $W_1 = (c_{r+1}, \ldots, c_{s-1})$ est un chemin simple et $\Omega_2 = (c_s, \ldots, c_{t-1})$ est un cycle simple qui remplit les deux propriétés du lemme. En continuant cet algorithme jusqu'à la fin de C, nous construisons la représentation souhaitée (notez que le dernier chemin simple ajouté à la représentation est W_q s'il commence par un sommet différent de c_1, sinon c'est Ω_q). □

Une décomposition d'un cycle respectant le Lemme 3 est appelée *décomposition cycle-chemin* de C.

Notre prochain objectif est de montrer l'existence de décompositions cycle-chemin avec une description compacte, c'est-à-dire avec un nombre réduit de chemins simples. La preuve se base sur le fait que, s'il y a trop de cycles différents dans la décomposition, alors certains d'entre eux peuvent être remplacés par des répétitions d'autres cycles. Avant de prouver cela par le Lemme 5, nous montrons l'exactitude de l'opération d'échange suivant.

Lemme 4. *Soit C un cycle couvrant dans un graphe G avec une décomposition cycle-chemin $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$. Si C contient un cycle Ω_j tel que :*

- *il existe un cycle Ω_i, $i < j$, qui a la même longueur que Ω_j, et*
- *pour chaque arc $a \in A(\Omega_j)$, il existe un cycle Ω_p, $p \neq j$, tel que $a \in A(\Omega_p)$,*

alors $C' = \Omega_1 \cdot \ldots \cdot W_{i-1} \cdot \Omega_i^2 \cdot W_i \cdot \ldots \cdot W_{j-1} \cdot W_j \cdot \ldots \cdot W_q$ est un cycle couvrant ayant la même longueur dans G.

Démonstration. Soient $\Omega_i = (u_1, u_2, \ldots, u_s)$ et $W_i \cdot \Omega_{i+1} = (w_1, w_2, \ldots, w_t)$. Comme Ω_i est un cycle, $\Omega_i \cdot \Omega_i \cdot W_i$ est un chemin. Nous considérons $\Omega_{j-1} \cdot W_{j-1} = (x_1, \ldots, x_s)$, $\Omega_j = (y_1, \ldots, y_t)$, et $W_j \cdot \Omega_{j+1} = (z_1, \ldots, z_r)$. Comme C est un cycle, on déduit $(x_s, y_1) \in A$ et, par les propriétés de la décomposition cycle-chemin, on déduit également $y_1 = z_1$. Par conséquence, $(x_s, z_1) \in A$ et $\Omega_{j-1} \cdot W_{j-1} \cdot W_j \cdot \Omega_{j+1}$ est un chemin. Donc, C' est un cycle. Comme Ω_i et Ω_j ont la même longueur, C' possède la même longueur que C. De plus, C' est couvrant comme chaque arc de Ω_j est contenu dans un chemin Ω_p, $p \neq j$. □

En utilisant l'opération d'échange décrite dans le Lemme 4, nous montrons maintenant qu'il existe des décompositions cycle-chemin compactes.

Lemme 5. *Si un graphe orienté $G = (V, A)$ a un cycle couvrant C de longueur ℓ, alors il possède un cycle couvrant C' de longueur ℓ tel que C' a une décomposition cycle-chemin $(\Omega_1)^{r_1} \cdot W_1 \cdot \ldots \cdot (\Omega_q)^{r_q} \cdot W_q$ où chaque r_i est un entier positif pour $1 \leq i \leq q$, $q \leq n+m$, $|V| = n$ et $|A| = m$.*

Démonstration. Nous supposons que G possède un cycle couvrant C de longueur ℓ. Selon le Lemme 3, C a une décomposition cycle-chemin $(\Omega_1)^{r_1} \cdot W_1 \cdot \ldots \cdot (\Omega_q)^{r_q} \cdot W_q$ (le Lemme 3 montre l'existence du cas particulier $r_1 = \ldots = r_q = 1$). Nous considérons maintenant que C est le cycle avec une décomposition dans laquelle $q + \sum_{i=1}^{q} |W_i|$ est minimum parmi tous les cycles couvrants de longueur ℓ de G.

Nous supposons, pour en déduire une contradiction, que dans cette décomposition, il y a des indices i et j, $i \neq j$, tels que $|\Omega_i| = |\Omega_j|$ et que chaque arc de Ω_j est contenu dans un cycle Ω_p, $p \neq j$. Sans perte de généralité, on considère $i < j$. Par la suite, nous transformons C en un nouveau cycle C' dans lequel $q + \sum_{i=1}^{q} |W_i|$ est plus petit, ce qui contredit notre choix de C.

Par la supposition pour i et j et par le Lemme 4, $C' = \Omega_1 \cdot \ldots \cdot W_{i-1} \cdot (\Omega_i)^{r_i+r_j} \cdot W_i \cdot \ldots \cdot \Omega_{j-1} \cdot W_{j-1} \cdot W_j \cdot \Omega_{j+1} \cdot \ldots \cdot W_q$ est aussi un cycle couvrant de G. Par conséquent, on a $|C| = |C'| = \ell$ et C' est aussi un cycle couvrant. Nous considérons maintenant deux cas.

Cas 1 : $W_{j-1} \cdot W_j$ contient un cycle simple Ω^*. Soit $W_{j-1} \cdot W_j = W_1^* \cdot \Omega^* \cdot W_2^*$ où W_1^* et W_2^* ne sont pas des cycles simples. Alors, $C' = \Omega_1 \cdot \ldots \cdot W_{i-1} \cdot (\Omega_i)^{r_i+r_j} \cdot W_i \cdot \ldots \cdot \Omega_{j-1} \cdot W_1^* \cdot \Omega^* \cdot W_2^* \cdot \Omega_{j+1} \cdot \ldots \cdot W_q$. Dans cette décomposition, le nombre total de Ω n'a pas changé mais, comme $|W_1^*| + |W_2^*| < |W_{j-1}| + |W_j|$, la somme des longueurs de tous les W_i a diminué. Ceci contredit notre choix pour C.

Cas 2 : Le cas opposé. Dans ce cas, $W_{j-1} \cdot W_j$ est un chemin simple. Par conséquent, $\sum_{i=1}^{q} |W_i|$ reste le même, tandis que le nombre de W et donc q a diminué de 1. Ceci contredit notre choix de C.

Comme les deux cas conduisent à une contradiction du choix pour C nous pouvons assumer, pour chaque Ω_j dans C, soit qu'il existe un arc a_j qui n'est contenu par aucun autre cycle Ω_p, $p \neq j$, soit qu'il n'y a pas d'autre cycle Ω_i de la même longueur que Ω_j. Par le principe des tiroirs, il peut exister au plus $|A| = m$ cycles Ω_j pour lesquels la première condition est vraie. Pour tous les autres cycles, la première condition est fausse. Comme chaque cycle Ω_j possède une longueur maximale de n, le principe des tiroirs nous permet, encore une fois, d'affirmer qu'il existe au plus n autres cycles pour lesquels la deuxième condition est vraie. Cela implique que $q \leq m + n$. □

Le lemme suivant montre que la longueur des cycles dans une décomposition suffit à déterminer la longueur totale du cycle.

Lemme 6. *Un graphe G possède un cycle couvrant C de longueur ℓ si et seulement s'il existe un cycle couvrant C' avec une décomposition cycle-chemin $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ tel que :*

1. *C' a une longueur $x \leq 2n(m+n)$, et*
2. *il existe les entiers non négatifs p_i, avec $1 \leq i \leq q$, tels que $x + \sum_{i=1}^{q} p_i \cdot |\Omega_i| = \ell$.*

Démonstration. Supposons que G possède un cycle couvrant C de longueur ℓ. Alors, par le Lemme 5, nous pouvons supposer que C a une décomposition cycle-chemin $\Omega_1^{r_1} \cdot W_1 \cdot \ldots \cdot \Omega_q^{r_q} \cdot W_q$ telle que $q \leq m + n$.

Nous considérons maintenant le cycle $C' = \Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$, c'est-à-dire le cycle dans lequel chaque cycle Ω_i est parcouru exactement une fois. Le cycle C' est donc couvrant. Comme $q \leq m + n$ et chaque Ω_i et W_i sont des chemins simples, C' a une longueur maximale de $2n(m + n)$. Par conséquent, C' respecte la première propriété du lemme. Nous notons x la longueur de C'. Alors,

$$\ell - x = \sum_{i=1}^{q} (r_i - 1) \cdot |\Omega_i|.$$

Ainsi, en mettant $p_i = r_i - 1$ pour chaque $i \in \{1, \ldots, q\}$, on démontre que la deuxième propriété est également satisfaite.

La réciproque de ce lemme peut être facilement démontrée en considérant le cycle qui est obtenu à partir de C' en ajoutant p_i parcours supplémentaires de chaque Ω_i.

□

Nous limitons maintenant le temps d'exécution pour déterminer l'existence d'un tel cycle.

Théorème 1. *Soit $G = (V, A)$ un graphe orienté avec n nœuds et m arcs, et soit ℓ un entier. Alors, on peut déterminer si G contient un cycle couvrant de longueur ℓ en un temps de $O(8^m \cdot 2^n) \cdot \mathrm{poly}(n + \log \ell)$.*

Démonstration. Par le Lemme 6 on peut résoudre le problème en déterminant s'il existe un cycle couvrant C avec une décomposition cycle-chemin qui respecte les deux propriétés du lemme. Nous résolvons ce problème en utilisant la programmation dynamique.

L'algorithme de programmation dynamique remplit un tableau $\mathcal{T}$ avec des entrées du type $\mathcal{T}[u, v, A', \Lambda, y]$ où u et v sont des nœuds de G, A' est un sous-ensemble de A, Λ est un sous-ensemble de $\{1, \ldots, n\}$ et y est un entier non négatif d'une valeur maximale de $2n \cdot (n + m)$.

Chaque entrée est vraie ou fausse. L'algorithme remplit le tableau $\mathcal{T}$ tel que chaque entrée $\mathcal{T}[u, v, A', \Lambda, y]$ est vraie si et seulement s'il existe un cycle C de u vers v avec une décomposition cycle-chemin $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ tel que :

- C contient exactement les arcs de A',
- $\Lambda = \{|\Omega_i| \mid 1 \leq i \leq q\}$, et
- C possède la longueur y.

L'idée est que, comme le montre le Lemme 6, pour déterminer si l'existence de C implique que G a un cycle couvrant de longueur ℓ, alors seulement les informations suivantes sont utiles : l'ensemble des arcs de C, la longueur de C, ainsi que la longueur des cycles dans sa décomposition cycle-chemin.

Nous décrivons tout d'abord comment remplir un tableau auxiliaire dans un prétraitement, puis comment remplir le tableau $\mathcal{T}$ et enfin comment utiliser $\mathcal{T}$ pour déterminer l'existence d'un cycle couvrant de la bonne longueur.

Prétraitement. On calcule un tableau $\mathcal{D}$ avec des entrées du type $\mathcal{D}[u, v, A', y]$ tel que $\mathcal{D}[u, v, A', y]$ est vrai si et seulement s'il existe dans G un chemin simple à partir de u vers v qui contient l'ensemble des arcs A' et de longueur y. Il faut noter que si l'ensemble des arcs A' contient deux arcs avec le même nœud entrant ou le même nœud sortant, il n'existe pas de chemin simple avec l'ensemble des arcs A'. En conséquence, toutes les entrées du tableau pour un tel ensemble A' sont marquées comme fausses. Pour les entrées restantes, le tableau $\mathcal{D}$ est rempli comme suit.

Nous calculons $\mathcal{D}$ pour des valeurs croissantes de y. Pour $y = 1$, on définit $\mathcal{D}[u, u, \emptyset, 1]$ vrai pour tout $u \in V$ et toutes les autres entrées du tableau comme fausses. Pour $y > 1$, on définit

$$\mathcal{D}[u, v, A', y] = \bigvee_{(w,v)\in A} \mathcal{D}[u, w, A' \setminus \{(w, v)\}, y - 1].$$

Il faut noter que s'il existe une entrée $\mathcal{D}[u, v, A', y]$ dans le tableau telle qu'il existe un arc (v, u) dans G, alors il existe un cycle $(u, \ldots, v)$ de longueur y qui contient exactement l'ensemble de l'arc A'. La réciproque est également vraie. S'il existe un tel cycle, l'entrée correspondante du tableau est vraie.

Remplissage de $\mathcal{T}$. L'étape suivante est de remplir le tableau $\mathcal{T}$ pour des valeurs croissantes de y. Nous supposons par la suite que les entrées du tableau pour les plus petites valeurs de y ont été remplies.

Pour calculer une entrée $\mathcal{T}[u, v, A', \Lambda, y]$ nous considérons plusieurs cas qui rendent l'entrée vraie. Dans le premier cas, le chemin est constitué seulement d'un cycle Ω_1. On peut utiliser directement le tableau $\mathcal{D}$ pour déterminer l'existence de ces chemins. Le deuxième cas est celui dans lequel le dernier arc (w, v) du chemin n'appartient pas à un chemin simple de la décomposition cycle-chemin. Dans ce cas, il existe un chemin de longueur $y-1$ finissant en w. En conséquence, l'entrée du tableau pour ce chemin doit être vraie (il faut noter que ce chemin peut éventuellement déjà contenir (w, v)). Le dernier cas est celui où le chemin se termine avec un cycle simple Ω_i de longueur z, $1 \leq z \leq n$. Dans ce cas, il existe un chemin, de longueur $y - z$, se terminant par un prédécesseur du premier nœud de Ω_i, et contenant tous les arcs dans A' qui ne sont pas dans Ω_i. Il faut noter que tous les candidats pour Ω_i peuvent être trouvés dans le tableau $\mathcal{D}$.

Suite à cette distinction de cas, la récurrence de remplissage du tableau $\mathcal{T}$ devient :

$$\mathcal{T}[u, v, A', \Lambda, y] = \begin{cases} \text{vrai} & \text{si } \Lambda = \{y\} \wedge \mathcal{D}[u, v, A', y] \wedge (v, u) \in A, \\ \text{vrai} & \text{si } \exists (w, v) \in A : \mathcal{T}[u, w, A', \Lambda, y - 1] \vee \\ & \qquad \mathcal{T}[u, w, A' \setminus \{(w, v)\}, \Lambda, y - 1], \\ \text{vrai} & \text{si } \exists \mathcal{T}[u, w, \tilde{A}, \tilde{\Lambda}, y - z], \mathcal{D}[w', v, A^*, z] : \\ & \mathcal{T}[u, w, \tilde{A}, \tilde{\Lambda}, y - z] \wedge \mathcal{D}[w', v, A^*, z] \wedge \\ & (w, w') \in A \wedge \tilde{A} \cup A^* \cup \{(w, w')\} = A' \wedge \\ & \tilde{\Lambda} \in \{\Lambda, \Lambda \setminus \{z\}\} \\ \text{faux} & \text{sinon.} \end{cases}.$$

Détermination de l'existence d'un cycle couvrant. Après le remplissage du tableau $\mathcal{T}$, il nous reste à calculer si une entrée du tableau implique l'existence d'un cycle couvrant C' qui respecte les propriétés du Lemme 6. Nous supposons qu'il existe un cycle C' ayant une décomposition cycle-chemin qui respecte ces propriétés. Soit y la longueur de C',

soit λ_i la longueur de Ω_i, et soit $\Lambda = \{\lambda_i \mid 1 \leq i \leq q\}$ l'ensemble des longueurs de tous les cycles Ω_i. Alors, l'entrée $\mathcal{T}[u, v, A \setminus \{(u, v)\}, \Lambda, y]$ ou l'entrée $\mathcal{T}[u, v, A, \Lambda, y]$ est vraie. De plus, il existe les entiers non négatifs $p_1, \ldots p_{|\Lambda|}$ tels que :

$$p_1\lambda_1 + p_2\lambda_2 + \ldots + p_{|\Lambda|}\lambda_{|\Lambda|} = \ell - y. \tag{6.2}$$

Inversement, s'il existe une entrée $\mathcal{T}[u, v, A \setminus \{(u, v)\}, \Lambda, y]$ ou $\mathcal{T}[u, v, A, \Lambda, y]$ qui satisfait cette propriété et si $(v, u) \in A$, alors il existe un cycle couvrant de longueur ℓ.

Ainsi, nous pouvons résoudre le problème initial en vérifiant pour chaque entrée du tableau si elle satisfait ces propriétés. La seule partie non triviale est de vérifier si l'équation (6.2) a une solution dans laquelle chaque p_i est un entier non négatif. Ce problème est connu sous le nom du PROBLÈME DU RENDU DE MONNAIE [9].

Analyse du temps d'exécution. Le temps d'exécution de l'algorithme est dominé par le temps nécessaire pour remplir le tableau $\mathcal{T}$. Ce tableau contient une entrée pour chaque paire de nœuds (u, v) (n^2 possibilités), pour chaque combinaison A' d'arcs de A (2^m possibilités), pour chaque combinaison Λ de longueur de cycles (2^n possibilités) et pour chaque longueur y telle que $1 \leq y \leq 2n(n + m)$. En conséquence, le tableau $\mathcal{T}$ contient un total de $n^2 \cdot (2^m) \cdot (2^n) \cdot 2n(n+m)$ entrées. Chaque entrée peut être calculée en prenant en compte une entrée dans $\mathcal{D}$ pour le premier cas de la récurrence, $2m$ entrées dans $\mathcal{T}$ (2 entrées pour chaque arc $(w, v) \in A$) pour le deuxième cas de la récurrence et $m \cdot n \cdot 4^m \cdot 2$ entrées dans $\mathcal{T}$ et $\mathcal{D}$ (une entrée pour chaque arc $(w, w') \in A$, pour chaque longueur possible z tel que $1 \leq z \leq n$, pour chaque combinaison d'arcs $\tilde{A}$ et A^*, et pour les deux valeurs possibles pour $\tilde{\Lambda}$) pour le troisième cas de la récurrence. Pour chaque possibilité, la vérification de la condition prend un temps de $O(m)$. Au total, le temps d'exécution pour cette partie de l'algorithme est donc de $O(8^m \cdot 2^n) \cdot \text{poly}(n)$.

Ensuite, nous devons résoudre une instance du PROBLÈME DU RENDU DE MONNAIE pour chaque paire de nœuds (u, v) (n^2 possibilités), pour chaque combinaison Λ de longueur de cycles (2^n possibilités) et pour chaque longueur y telle que $1 \leq y \leq 2n(n+m)$. En conséquence, nous devons résoudre $O(n^2 \cdot 2^n \cdot n(n + m))$ d'instances du PROBLÈME DU RENDU DE MONNAIE. Le PROBLÈME DU RENDU DE MONNAIE peut être résolu en un temps de $w \cdot N \cdot \text{poly}(\log X)$ [9]. Ici, w désigne le plus grand nombre entier d'entrées sur le côté gauche de l'équation (6.2) (dans notre cas, c'est le plus grand λ_i et donc au plus n), N désigne le nombre de numéros d'entrée (dans notre cas $|\Lambda| \leq n$), et X désigne le plus grand nombre entier dans l'entrée (dans notre cas ℓ). Par conséquent, chaque instance du PROBLÈME DU RENDU DE MONNAIE peut être résolu en un temps

$$n \cdot n \cdot \text{poly}(\log \ell) = n^2 \cdot \text{poly}(\log \ell).$$

Ainsi, nous avons besoin de $O(2^n) \cdot \text{poly}(n + \log \ell)$ pour cette partie de l'instance. □

6.4 Algorithme pour le PROBLÈME DU CYCLE COUVRANT SOLIDE pour les paramètres n et d

Nous décrivons maintenant notre premier algorithme pour le PROBLÈME DU CYCLE COUVRANT SOLIDE. Le temps d'exécution de l'algorithme est exponentiel seulement en n et d. Ainsi, nous évitons une complexité exponentielle du nombre ℓ qui est au moins aussi grand que d et généralement beaucoup plus grand.

L'algorithme exploite le fait que, dans un chemin solide, des parties avec une distance plus grande que d sont "indépendantes" par rapport à la propriété de solidité. Pour rendre l'argument plus précis, nous considérons une instance (G, d, ℓ) du PROBLÈME DU CYCLE COUVRANT SOLIDE qui a comme solution le cycle $C = W_1 \cdot W_2 \cdot \ldots \cdot W_q \cdot W^*$, où les chemins $|W_i| = d$ pour $1 \leq i \leq q$ et $|W^*| = \ell \mod d$. Pour chaque nœud v_j dans W_i, le nœud qui est nécessaire pour déterminer si v_j est solide est contenu dans W_{i+1}. Ainsi, nous considérons le graphe $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ qui contient chaque chemin de G de longueur d comme nœud. En particulier, $\mathcal{G}$ contient chaque W_i. De plus, nous supposons que $\mathcal{G}$ contient un arc (W, W') si $W \cdot W'$ est un chemin dans G qui est solide pour toutes les positions dans W. Alors $(W_i, W_{i+1}) \in \mathcal{A}$ pour chaque $i < q$. En conséquence, le chemin $W_1 \cdot W_2 \cdot \ldots \cdot W_q$ dans G correspond à un chemin $\mathcal{W}$ dans $\mathcal{G}$ et $W_1 \cdot W_2 \cdot \ldots \cdot W_q \cdot W^*$ et est un cycle couvrant solide de longueur ℓ dans G.

Les étapes de l'algorithme sont donc les suivantes. Tout d'abord, on construit le graphe $\mathcal{G}$, appelé *graphe des chemins* par la suite. La deuxième étape consiste en un calcul des chemins "candidats" dans $\mathcal{G}$. La troisième et dernière étape est la vérification pour chaque chemin candidat, s'il existe un chemin court W^* tel que la concaténation de W^* à l'extrémité du chemin candidat représente un cycle couvrant solide avec la longueur nécessaire.

Théorème 2. *Le* PROBLÈME DU CYCLE COUVRANT SOLIDE *peut être résolu dans* $O(8^{n \cdot \Delta} \cdot 2^{n \cdot \Delta^d}) \cdot \mathrm{poly}(n \cdot \Delta^d + \log \ell)$ *où* Δ *est le degré sortant maximum de* G.

Démonstration. Nous décrivons chacune des trois principales étapes de l'algorithme en détail et ensuite nous calculons sa complexité.

Construction du graphe des chemins $\mathcal{G}$. Tout d'abord, on énumère tous les chemins de longueur d dans G. Soit $\mathcal{V}$ l'ensemble de ces chemins qui représente également l'ensemble des nœuds du graphe $\mathcal{G}$. Ensuite, on construit l'ensemble des arcs $\mathcal{A}$ de $\mathcal{G}$ comme suit. Pour chaque paire de nœuds W et W' dans $\mathcal{V}$, on vérifie si $W \cdot W' = (v_1, \ldots, v_{2d})$ est un chemin dans G et s'il est solide pour chaque v_i, $1 \leq i \leq d$. Pour cela, on vérifie si $(v_d, v_{d+1}) \in A$ et si $\mathcal{R}(v_i) = \mathcal{L}(v_{i+d})$ pour chaque v_i, $1 \leq i \leq d$. Si c'est le cas, on ajoute l'arc (W, W') au graphe $\mathcal{G}$, sinon, on ne construit pas cet arc. Cela achève la construction du graphe de chemins $\mathcal{G}$. Des chemins quasi-solides (c'est-à-dire des chemins solides pour certaines positions) dans G correspondent à des chemins dans $\mathcal{G}$.

> *Observation 1 :* Un chemin $W_1 \cdot \ldots \cdot W_i$ de longueur $d \cdot i$ dans G avec $|W_j| = d$, $1 \leq j \leq i$, est solide pour ses $d \cdot (i-1)$ premières positions $\Leftrightarrow$ $(W_1, \ldots, W_i)$ est un chemin dans $\mathcal{G}$.

Prétraitement. Pour un chemin $(W_1, \ldots, W_i)$ dans $\mathcal{G}$, soit $A(W_1, \ldots, W_i) = A(W_1) \cup A(W_2) \cup \ldots \cup A(W_i)$ l'ensemble des arcs de $W_1 \cdot \ldots \cdot W_i$ dans G. De plus, pour un arc (W, W') dans $\mathcal{G}$ avec $W = (v_1, \ldots, v_d)$ et $W' = (v_{d+1}, \ldots, v_{2d})$, on note $\mathrm{arc}(W, W')$ l'arc (v_d, v_{d+1}) dans G (par la construction du graphe des chemins, cet arc est présent dans G).

Nous résolvons maintenant le PROBLÈME DU CYCLE COUVRANT SOLIDE en déterminant l'existence d'un chemin $(W_1, \ldots, W_q)$ de longueur $q = \lfloor \ell / d \rfloor$ dans $\mathcal{G}$ et d'un chemin W^* de longueur $\ell \mod d$ dans G tel que :

- $W_q \cdot W^* \cdot W_1$ est un chemin de longueur $2d + (\ell \mod d)$ dans G qui est solide pour ses $d + (\ell \mod d)$ premières positions, et

- chaque arc de G est contenu dans $A(W_1, \ldots, W_q)$ ou dans $W_q \cdot W^* \cdot W_1$.

Pour cela, nous utilisons un algorithme de programmation dynamique qui remplit un tableau $\mathcal{T}$ avec des entrées de type $\mathcal{T}[W, W', A', \Lambda, y]$ où :

- W et W' sont des nœuds de $\mathcal{G}$,
- A' est un sous-ensemble de A (il faut noter que A est l'ensemble des arcs de G et pas de $\mathcal{G}$),
- Λ est un sous-ensemble de $\{1, \ldots, |\mathcal{V}|\}$, et
- y est un entier non négatif d'une valeur maximale de $2|\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{A}|)$.

Chaque entrée dans $\mathcal{T}$ est, soit vraie, soit fausse. Le but de l'algorithme est de remplir le tableau $\mathcal{T}$ tel que $\mathcal{T}[W, W', A', \Lambda, y]$ est vrai si et seulement si $\mathcal{G}$ contient un chemin $(W, \ldots, W')$ avec une décomposition cycle-chemin $\Omega_1 \cdot \Psi_1 \cdot \ldots \cdot \Omega_i \cdot \Psi_i$ telle que :

- $A(W, \ldots, W') = A'$, c'est-à-dire, le chemin $W \cdot \ldots \cdot W'$ dans G contient exactement les arcs de A',
- $\Lambda = \{|\Omega_j| \mid 1 \leq j \leq i\}$, et
- $(W, \ldots, W')$ possède une longueur de y.

Le tableau $\mathcal{T}$ est construit pour que, par la suite, en appliquant le Lemme 6, on puisse trouver le cycle couvrant solide en considérant uniquement des chemins de longueur maximale $2|\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{A}|)$ qui seront étendus en utilisant les longueurs de cycle dans Λ. Dans une étape de prétraitement, on calcule un tableau $\mathcal{D}$. Pour le tableau $\mathcal{D}$ correctement rempli, une entrée $\mathcal{D}[W, W', A', y]$ est vraie si et seulement si $\mathcal{G}$ contient un chemin $(W, \ldots, W')$ de longueur y tel que $A(W, \ldots, W') = A'$. Le tableau est rempli pour tout $A' \subseteq A$ et pour des valeurs croissantes de $y < |\mathcal{V}|$. Initialement, on positionne $\mathcal{D}[W, W, A(W), 1]$ à vrai pour chaque $W \in \mathcal{V}$. Ensuite, la récurrence pour $\mathcal{D}$ est :

$$\mathcal{D}[W, W', A', y] = \begin{cases} \text{vrai} & \text{si } \exists (\tilde{W}, W') \in \mathcal{A}, \tilde{A} \subseteq A' : \\ & \quad \tilde{A} \cup \{\text{arc}(\tilde{W}, W')\} \cup A(W') = A' \wedge \\ & \quad \mathcal{D}[W, \tilde{W}, \tilde{A}, y - 1], \\ \text{faux} & \text{sinon.} \end{cases}$$

Une fois que le tableau $\mathcal{D}$ est complètement rempli, on calcule le tableau $\mathcal{T}$. La récurrence est :

$$\mathcal{T}[W, W', A', \Lambda, y] = \begin{cases} \text{vrai} & \text{si } \Lambda = \{y\} \wedge \mathcal{D}[W, W', A', y] \wedge (W', W) \in \mathcal{A}, \\ \text{vrai} & \text{si } \exists (\tilde{W}, W') \in \mathcal{A}, \tilde{A} \subseteq A' : \\ & \quad \tilde{A} \cup \{\text{arc}(\tilde{W}, W')\} \cup A(W') = A' \wedge \\ & \quad \mathcal{T}[W, \tilde{W}, \tilde{A}, \Lambda, y - 1], \\ \text{vrai} & \text{si } \exists \mathcal{T}[W, \tilde{W}, \tilde{A}, \tilde{\Lambda}, y - z], \mathcal{D}[\tilde{W}, W', A^*, z] : \\ & \quad \mathcal{T}[W, \tilde{W}, \tilde{A}, \tilde{\Lambda}, y - z] \wedge \mathcal{D}[\hat{W}, W', A^*, z] \wedge \\ & \quad (\tilde{W}, \hat{W}) \in \mathcal{A} \wedge \tilde{A} \cup \{\text{arc}(\tilde{W}, \hat{W})\} \cup A^* = A' \wedge \\ & \quad \Lambda \setminus \{z\} \subseteq \tilde{\Lambda} \subseteq \Lambda, \\ \text{faux} & \text{sinon.} \end{cases}$$

Détermination des longueurs possibles pour les entrées candidates. L'étape suivante consiste à calculer pour chaque entrée si elle peut être étendue, par des cycles répétés de la décomposition cycle-chemin, pour obtenir un chemin de longueur $\ell - (\ell \mod d)$.

Comme dans la démonstration du Théorème 1, ceci est réalisé par une réduction au PROBLÈME DU RENDU DE MONNAIE [9]. Plus précisément, pour chaque entrée vraie $\mathcal{T}[W, W', A', \Lambda, y]$, on vérifie s'il existe un chemin de longueur $\ell - (\ell \mod d)$ qui peut être obtenu par une répétition des cycles simples, pour ensuite former un chemin $\mathcal{W}$ de longueur y dont l'existence est impliquée par l'entrée du tableau. L'ensemble des longueurs différentes des cycles simples dans $\mathcal{W}$ est $\Lambda = \{\lambda_1, \ldots, \lambda_{|\Lambda|}\}$. Ainsi, la détermination de l'existence d'un cycle de longueur $\ell - (\ell \mod d)$ est équivalente à la vérification que l'équation

$$p_1\lambda_1 + p_2\lambda_2 + \ldots + p_{|\Lambda|}\lambda_{|\Lambda|} = \ell - (\ell \mod d) - y$$

a une solution dans laquelle chaque p_i est un entier non négatif. Chaque entrée du tableau pour laquelle l'équation ci-dessus a une solution est identifiée comme étant une *entrée candidate*. Par le Lemme 6, l'existence d'un chemin de longueur $\ell - (\ell \mod d)$ implique qu'il existe une entrée candidate correspondante. Ainsi, si $\mathcal{T}$ ne contient aucune entrée candidate après cette étape, alors l'instance ne possède pas de solution.

Fermeture du cycle. La dernière étape de l'algorithme est de vérifier si l'une des entrées candidates peut être complétée pour former un cycle solide de longueur ℓ en ajoutant un chemin de longueur $\ell \mod d$. Pour cela, tout d'abord, on énumère tous les chemins W^* de longueur $\ell \mod d$ dans G selon Lemme 1. Ensuite, pour chaque entrée candidate $\mathcal{T}[W, W', A', \Lambda, y]$ et pour chaque chemin W^* énuméré, on procède comme suit. On vérifie si $W' \cdot W^* \cdot W$ est un chemin dans G. Si oui, alors $W \cdot \ldots \cdot W' \cdot W^*$ correspond à un cycle C dans G. Ce cycle a une longueur de $y + (\ell \mod d)$ mais comme on ne considère que les entrées candidates dans $\mathcal{T}$, le chemin $W \cdot \ldots \cdot W'$ de longueur y peut être étendu pour former un chemin de longueur $\ell - (\ell \mod d)$. Ainsi, l'existence de C implique l'existence d'un cycle C' de longueur ℓ dans G. Il nous reste à vérifier si C' est solide et couvrant. Pour vérifier si C' est solide, il suffit de vérifier si le chemin $W' \cdot W^* \cdot W$ est solide pour ses $d + (\ell \mod d)$ premières positions (comme chaque chemin dans $\mathcal{G}$ correspond à un chemin dans G dont toutes les positions sont solides, sauf pour les d dernières). Enfin, il nous reste à vérifier si C' est couvrant. Cela se fait en vérifiant si $A = A' \cup \{(w'_d, w^*_1), (w^*_{\ell \mod d}, w_1)\} \cup A(W^*)$, où w'_d est le dernier nœud de W', w^*_1 et $w^*_{\ell \mod d}$ sont le premier et respectivement le dernier nœud de W^* et w_1 est le premier nœud de W dans G. Si oui, C' est une solution. Si aucune des combinaisons d'entrée candidate et W^* ne fournit de solution, alors l'instance n'a pas de solution.

Analyse du temps d'exécution. Le Lemme 1 nous indique que le nombre de chemins différents de longueur d dans G est d'au maximum $n \cdot \Delta^{d-1}$ et donc $|\mathcal{V}| \leq n \cdot \Delta^{d-1}$. En conséquence, la construction de $\mathcal{G}$ peut être réalisée en un temps de $\mathrm{poly}(n \cdot \Delta^{d-1})$.

Le temps d'exécution dans la partie de programmation dynamique est dominé par le temps de remplissage du tableau $\mathcal{T}$. Cela est dominé à son tour par le temps nécessaire pour vérifier si le troisième cas de la récursion s'applique. Pour ce faire, on doit considérer, pour chaque entrée du tableau, $O(|\mathcal{V}|^2 \cdot 4^{n \cdot \Delta})$ possibilités on rappelle que $n \cdot \Delta \geq |A|$). Pour chaque possibilité, la vérification peut être réalisée en $\mathrm{poly}(|\mathcal{V}|)$.

Le tableau $\mathcal{T}$ contient une entrée pour chaque paire de nœuds (W, W') ($|\mathcal{V}|^2$ possibilités), pour chaque combinaison A' d'arcs de A ($2^{n \cdot \Delta}$ possibilités), pour chaque com-

binaison Λ de longueur de cycles ($2^{|\mathcal{V}|}$ possibilités) et pour chaque longueur y telle que $1 \leq y \leq 2|\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{E}|)$. En conséquence, il y a $O(|\mathcal{V}|^2 \cdot 2^{n\cdot\Delta} \cdot 2^{|\mathcal{V}|} \cdot |\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{E}|)) = 2^{n\cdot\Delta} \cdot 2^{|\mathcal{V}|} \cdot \mathrm{poly}(|\mathcal{V}|)$ entrées du tableau à calculer. Ainsi, le temps d'exécution global dans la partie de la programmation dynamique est de $8^{n\cdot\Delta} \cdot 2^{|\mathcal{V}|} \cdot \mathrm{poly}(|\mathcal{V}|)$.

Ensuite, on résout $O(2^{n\cdot\Delta} \cdot 2^{|\mathcal{V}|})$ instances du PROBLÈME DU RENDU DE MONNAIE, chacune en un temps de $\mathrm{poly}(|\mathcal{V}| + \log \ell)$ [9]. Les vérifications de l'étape finale exigent un temps de $\mathrm{poly}(|\mathcal{V}|)$ pour chaque entrée candidate car moins de $|\mathcal{V}|$ chemins courts différents sont considérés, et chaque vérification nécessite un temps de $\mathrm{poly}(|\mathcal{V}|)$.

□

6.5 Algorithme de recherche du plus court cycle couvrant solide

Le principe décrit ci-dessus peut être utilisée dans plusieurs variantes du PROBLÈME DU CYCLE COUVRANT SOLIDE. Une possibilité consiste à trouver le plus court cycle couvrant solide à la place d'un cycle couvrant solide de longueur fixe.

PROBLÈME DU PLUS COURT CYCLE COUVRANT SOLIDE
Entrée : Un graphe de de Bruijn pairé $G = (V, A)$ et un entier non négatif d.
Objectif : Trouver le plus court cycle couvrant solide dans G.

Nous commençons par borner la longueur du plus court cycle couvrant solide dans G.

Lemme 7. *Soit $G = (V, A)$ un graphe orienté avec un degré sortant maximum Δ. Si G a un cycle couvrant solide, alors il a un cycle couvrant de longueur maximum $2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}) + d$.*

Démonstration. Nous considérons le graphe des chemins $\mathcal{G}$ de G. Ce graphe possède $n \cdot \Delta^{d-1}$ nœuds. Nous considérons également le chemin $\mathcal{W}$ dans $\mathcal{G}$ tel que $\mathcal{W} \cdot W^*$ est un cycle couvrant solide dans G, où $|W^*| < d$. Nous supposons que $\mathcal{W}$ est le plus court chemin avec cette propriété.

Soit $\Omega_1 \cdot W_1 \cdot \ldots \cdot \Omega_q \cdot W_q$ la décomposition cycle-chemin de $\mathcal{W}$ qui existe grâce au Lemme 3. Alors, pour chaque Ω_i, avec $i < q$, il existe un $a \in A(\Omega_i)$ qui n'est contenu dans aucun Ω_j, $j \neq i$. Sinon, en enlevant Ω_i de $\mathcal{W}$, on construit un chemin plus court dans $\mathcal{G}$ pour lequel le chemin correspondant dans G couvre également tous les arcs. Par le principe des tiroirs, cela implique $q \leq |A| + 1$. Ainsi, la longueur de $\mathcal{W}$ dans $\mathcal{G}$ est d'au maximum $2(|A| + 1) \cdot (n \cdot \Delta^{d-1})$ comme chaque Ω_i et chaque W_i sont des chemins simples dans $\mathcal{G}$. Le chemin correspondant dans G a une longueur maximale de $2d(|A| + 1) \cdot (n \cdot \Delta^{d-1})$ car chaque nœud de $\mathcal{G}$ correspond à un chemin de longueur d dans G. La limite globale est déduite du fait que $|A| \leq n \cdot \Delta$ et $|W^*| < d$. □

En utilisant cette borne, nous présentons maintenant un algorithme de programmation dynamique qui calcule des chemins de longueur croissante dans G. Dans ce calcul, nous enregistrons les derniers d nœuds du chemin, car ils possèdent une influence sur la condition de solidité. De plus, nous enregistrons les arcs de G qui sont déjà couverts par le chemin.

Théorème 3. *Le* PROBLÈME DU PLUS COURT CYCLE COUVRANT SOLIDE *peut être résolu en un temps de $O(2^{n\cdot\Delta} \cdot n^5 \cdot d \cdot \Delta^{3d})$.*

Démonstration. La complexité est évidemment maintenue si le plus court cycle couvrant solide a une longueur maximale de $n \cdot \Delta^{2d}$ (car une simple énumération décrite dans le Lemme 1 résout le problème dans ce cas). Ainsi, nous supposons par la suite qu'un plus court cycle couvrant solide C de G a une longueur minimum de $n \cdot \Delta^{2d}$. Nous commençons par l'énumération de tous les préfixes W possibles de C de longueur d. Nous remplissons ensuite un tableau $\mathcal{T}$ avec des entrées booléennes et des indices du type $\mathcal{T}[P, S, A', y]$. Dans le tableau complet, chaque entrée est vraie si et seulement si :

- il existe un chemin W dans G tel que son préfixe de longueur d est le chemin P,
- son suffixe de longueur d est exactement le chemin S,
- W est solide pour chaque position exceptant les d dernières positions,
- W contient exactement les arcs de A', et
- W a une longueur y avec $d \leq y \leq 2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}) + d$.

Le tableau est initialisé en mettant $\mathcal{T}[W, W, A(W), d]$ à vrai pour tous les chemins de longueur d dans G. Ensuite, les entrées du tableau sont calculées pour des valeurs croissantes de y. La récurrence est la suivante :

$$\mathcal{T}[P, (s_1, \ldots, s_d), A', y] = \begin{cases} \text{vrai} & \text{si} \quad \exists s_0 \in V, \tilde{A} \subseteq A' : \\ & \quad \mathcal{T}[P, (s_0, \ldots, s_{d-1}), \tilde{A}, y-1] \wedge \\ & \quad (s_{d-1}, s_d) \in A \wedge (\mathcal{R}(s_0) = \mathcal{L}(s_d)) \wedge \\ & \quad A' \setminus \{(s_{d-1}, s_d)\} \subseteq \tilde{A}, \\ \text{faux} & \text{sinon.} \end{cases}$$

Pendant le remplissage du tableau $\mathcal{T}$, on vérifie si chaque entrée $\mathcal{T}[P, (s_1, \ldots, s_d), A', y]$ vraie correspond à un cycle couvrant solide dans G. Pour ce faire, on doit seulement vérifier si :

- $S \cdot P$ est un chemin, cela implique que le chemin pour cette entrée du tableau est un cycle
- $S \cdot P$ est solide pour ses d premières positions, cela implique que le cycle est solide
- $A' \cup \{(s_d, s_1)\} = A$, cela implique que le cycle est couvrant.

Si cela est le cas, alors on renvoie le chemin trouvé.

Si ce test échoue pour chaque entrée calculée, alors l'instance n'a pas de solution. L'exactitude de l'algorithme découle du fait que le tableau est rempli correctement pour des chemins de longueur croissante et que, pour déterminer si un chemin est une solution, il suffit de considérer seulement le suffixe et préfixe de longueur d puisque toutes les positions exceptant les d dernières sont solides.

Nous limitons maintenant le temps d'exécution de l'algorithme. Le tableau $\mathcal{T}$ contient une entrée pour chaque paire de chemins (P, S) ($(n \cdot \Delta^{d-1})^2$ possibilités), pour chaque combinaison A' d'arcs de A ($2^{n \cdot \Delta}$ possibilités) et pour chaque longueur y telle que $d \leq y \leq 2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}) + d$. En conséquence, $\mathcal{T}$ contient $O(n^2 \cdot \Delta^{2d-2} \cdot 2^{n \cdot \Delta} \cdot 2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}))$ entrées en total. Pour calculer la valeur de chaque entrée, on doit considérer au plus n possibilités (pour chaque nœud $s_0 \in V$). Ainsi, le temps nécessaire pour remplir le tableau est en $O(2^{n \cdot \Delta} \cdot n^5 \cdot d \cdot \Delta^{3d})$. Le temps nécessaire pour vérifier si chaque entrée correspond à une solution est en $O(d + n)$. □

6.6 Algorithme de recherche du cycle couvrant solide avec une taille d'insert approximative

Pour la deuxième variante, nous relâchons la contrainte de la solidité. Ce choix est motivé par le fait que la taille d'insert des reads pairés n'est pas toujours égale à d bp. Cette notion relâchée de la solidité est définie comme suit.

Nous rappelons qu'un cycle dans un graphe de de Bruijn pairé correspond à deux séquences s et t. Un cycle C de longueur ℓ est appelé *x-approximativement solide* si

$$\forall i \in \{1 \leq i \leq \ell\} : t[i \mod l] \in \{s[i+d-x \mod l], \ldots, s[i+d \mod l]\}.$$

De manière informelle, cela signifie que pour un nœud v, $\mathcal{L}(v)$ est situé à une distance minimale $d-x$ et une distance maximale d de $\mathcal{R}(v)$. Cette définition conduit au problème suivant.

> PROBLÈME DU CYCLE COUVRANT SOLIDE APPROXIMATIF
> **Entrée :** Un graphe de de Bruijn pairé $G = (V, A)$ et les entiers non négatifs d, x et ℓ.
> **Question :** Le graphe G contient-il un cycle couvrant x-approximativement solide de longueur ℓ ?

Nous modifions légèrement l'algorithme du PROBLÈME DU CYCLE COUVRANT SOLIDE pour obtenir l'algorithme suivant :

Théorème 4. *Le* PROBLÈME DU CYCLE COUVRANT SOLIDE APPROXIMATIF *peut être résolu en* $O(8^{n\cdot\Delta} \cdot 2^{n\cdot\Delta^d}) \cdot \mathrm{poly}(n \cdot \Delta^d + \log \ell)$.

Démonstration. L'algorithme est presque le même que l'algorithme pour le PROBLÈME DU CYCLE COUVRANT SOLIDE décrit dans la démonstration du Théorème 2. Il y a seulement les deux différences. Premièrement, dans la construction du graphe des chemins $\mathcal{G}$, on ajoute un arc (W, W') si le chemin $W \cdot W'$ est x-approximativement solide pour ses d premières positions ; ensuite, chaque chemin dans $\mathcal{G}$ correspond a un chemin dans G qui est approximativement solide pour toutes ses positions sauf ses d dernières. Deuxièmement, quand on referme un cycle, on vérifie pour chaque entrée candidate $\mathcal{T}[W, W', A', \Lambda, y]$ et chaque W^*, si $W' \cdot W^* \cdot W$ est un chemin qui est approximativement solide pour ses $d + (\ell \mod d)$ premières positions. C'est le cas si et seulement si le cycle correspondant est approximativement solide pour toutes ses positions.

Le temps d'exécution de l'algorithme résultant est le même à une seule exception : tester si un chemin est approximativement solide pour ses $O(d)$ premières positions peut prendre $O(d^2)$ temps si réalisé d'une manière naïve. Cela ajoute seulement un facteur supplémentaire de $O(d)$ pour le temps d'exécution, donc la borne pour le temps d'exécution global reste la même. □

Enfin, nous considérons le problème de recherche du plus court cycle approximativement solide. Dans cette variante, on permet un nombre y de substitutions. C'est-à-dire, il peut exister y positions qui ne sont pas approximativement solides. Plus formellement, un chemin W de longueur ℓ est appelé *x-approximativement solide avec un coût maximal de* y s'il existe un ensemble $M \subseteq \{1, \ldots, \ell\}$ de cardinalité maximale y tel que

$$\forall i \in \{1, \ldots, \ell\} \setminus M : t[i \mod \ell] \in \{s[i+d-x \mod \ell], \ldots, s[i+d \mod \ell]\}.$$

Le PROBLÈME DU PLUS COURT CYCLE COUVRANT APPROXIMATIVEMENT SOLIDE LIMITÉ PAR LE COÛT consiste donc à trouver le cycle couvrant le plus court qui est x-approximativement solide avec un coût maximal y.

Pour obtenir un algorithme pour cette variante, il faut noter que le Lemme 7 est également valable pour le plus court cycle approximativement solide avec un coût limité. L'argument remplacé dans la preuve ne supprime que les cycles dans le graphe des chemins. Cette modification n'augmente pas le coût de la solution.

Théorème 5. *Le* PROBLÈME DU PLUS COURT CYCLE COUVRANT APPROXIMATIVEMENT SOLIDE LIMITÉ PAR LE COÛT *peut être résolu en* $O(2^{n\cdot\Delta} \cdot n^5 \cdot d^2 \cdot \Delta^{3d})$.

Démonstration. La complexité est évidemment maintenue si le plus court cycle couvrant solide a une longueur maximale de $n \cdot \Delta^{2d}$ car une simple énumération décrite dans le Lemme 1 résout le problème dans ce cas. Ainsi, on suppose, par la suite, que le plus court cycle C de G représentant la solution du problème a une longueur minimum de $n \cdot \Delta^{2d}$. Premièrement, on énumère tous les préfixes W possibles de longueur d de C. Ensuite, on remplit un tableau $\mathcal{T}$ avec des entrées entières et des indices de type $\mathcal{T}[P, S, A', y]$. Dans le tableau rempli entièrement, chaque entrée possède la valeur t si et seulement si :

- il existe un chemin W dans G tel que son préfixe de longueur d est le chemin P,
- son suffixe de longueur d est le chemin S,
- W est x-approximativement solide pour chaque position sauf pour t des $|W| - d$ premières positions,
- W contient exactement les arcs de A', et
- W a une longueur y, $d \leq y \leq 2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}) + d$.

Le tableau est initialisé en mettant $\mathcal{T}[W, W, A(W), d] = 0$ pour chaque chemin de longueur d dans G. Ensuite, les entrées du tableau sont calculées pour des valeurs croissantes de y. La récurrence est la suivante :

$$\mathcal{T}[P, (s_1, \ldots, s_d), A', y] =$$
$$\min_{s_0 \in V, \tilde{A} \subseteq A'} \begin{cases} +\infty & \text{si } (s_{d-1}, s_d) \notin A \\ +\infty & \text{si } \neg A' \setminus \{(s_{d-1}, s_d)\} \subseteq \tilde{A} \\ \mathcal{T}[P, (s_0, \ldots, s_{d-1}), \tilde{A}, y-1] & \text{si } \mathcal{R}(s_0) \in \{\mathcal{L}(s_{d-x}), \ldots, \mathcal{L}(s_d)\} \\ \mathcal{T}[P, (s_0, \ldots, s_{d-1}), \tilde{A}, y-1] + 1 & \text{sinon.} \end{cases}$$

Pendant le remplissage du tableau $\mathcal{T}$, on vérifie pour chaque $\mathcal{T}[P, (s_1, \ldots, s_d), A', y]$ si elle correspond à un cycle solution dans G. Pour ce faire, on doit vérifier si $S \cdot P$ est un chemin (cela implique que le chemin pour cette entrée du tableau est un cycle), si $A' \cup \{(s_d, s_1)\} = A$ (cela implique que le cycle est couvrant) et si le nombre de substitutions parmi ses d premières positions dans $S \cdot P$ plus $\mathcal{T}[P, (s_1, \ldots, s_d), A', y]$ est au maximum t. Si c'est le cas, alors on renvoie le cycle trouvé.

Si ce test échoue pour chaque entrée calculée, alors l'instance n'a pas de solution. L'exactitude de l'algorithme découle du fait que le tableau est rempli correctement pour des chemins de longueur croissante et que, pour déterminer si un chemin est une solution, il suffit de considérer seulement le suffixe et le préfixe ainsi que les coûts des chemins correspondant aux entrées du tableau.

Nous calculons maintenant le temps d'exécution de l'algorithme. Le tableau $\mathcal{T}$ contient une entrée pour chaque paire de chemins (P, S) ($(n \cdot \Delta^{d-1})^2$ possibilités), pour chaque combinaison A' d'arcs de A ($2^{n \cdot \Delta}$ possibilités) et pour chaque longueur y telle que $d \leq y \leq 2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}) + d$. En conséquence, $\mathcal{T}$ a $O(n^2 \cdot \Delta^{2d-2} \cdot 2^{n \cdot \Delta} \cdot 2d(n \cdot \Delta + 1) \cdot (n \cdot \Delta^{d-1}))$ entrées en total. Pour calculer la valeur de chaque entrée, on doit considérer au plus $O(n)$ possibilités (pour chaque nœud $s_0 \in V$). Pour chaque possibilité, les conditions pour les deux premiers cas peuvent être vérifiées en $O(1)$ et la condition pour le troisième cas peut être vérifiée en $O(d)$. Ainsi, le temps nécessaire pour remplir le tableau est en $O(2^{n \cdot \Delta} \cdot n^5 \cdot d^2 \cdot \Delta^{3d})$. Le temps nécessaire pour vérifier si chaque entrée correspond à une solution est en $O(d^2 + n)$. □

6.7 Cas spécial pour le paramètre n

Dans cette section, nous présentons un algorithme $O(f(n) \cdot \text{poly}(\log \ell))$ pour un cas spécial du PROBLÈME DU CYCLE COUVRANT SOLIDE. Pour décrire la structure de ce cas particulier, nous introduisons la notion suivante : le *graphe de compatibilité* pour un graphe de de Bruijn pairé $G = (V, A)$ est un graphe $H = (V, B)$ tel que $(a, b) \in B$ si $\mathcal{L}(a) = \mathcal{R}(b)$. Un exemple est détaillé Figure 6.2.

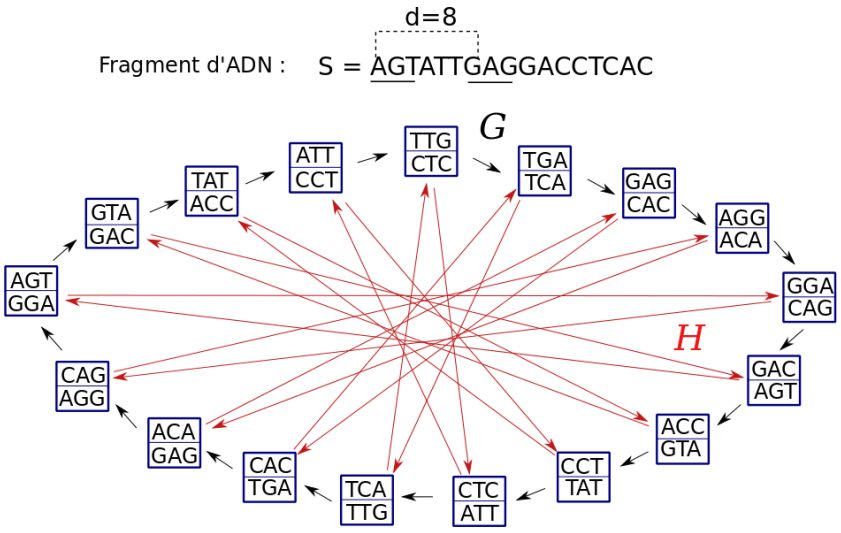


FIGURE 6.2 – Exemple du graphe de compatibilité H du graphe de de Bruijn pairé dans la Figure 6.1 : les arcs *noirs* appartiennent au graphe de de Bruijn pairé G, alors que les arcs *rouges* appartiennent au graphe de compatibilité H.

Nous exploitons maintenant cette structure en présentant un algorithme pour le cas où $H = (V, B)$ est une union de boucles, c'est-à-dire $B = \{(a, a) | \forall a \in V\}$. Dans ce cas, chaque paire de sommets avec une distance d dans un cycle solide est identique. En raison de ce comportement périodique, nous obtenons la relation suivante entre les cycles solides et des cycles couvrants plus courts.

La notation $pgcd(d, \ell)$ désigne le plus grand commun diviseur de d et ℓ.

Lemme 8. *Soit G un graphe de de Bruijn pairé tel que le graphe de compatibilité H soit une union de boucles. Alors, G a un cycle couvrant solide de longueur ℓ avec le décalage d si et seulement si G a un cycle couvrant de longueur $pgcd(d, \ell)$.*

Démonstration. Dans la démonstration, nous utilisons le Lemme de périodicité [37] qui est énoncé comme suit.

Lemme 9 ([37])**.** *Si p et q sont deux périodes d'un mot x telles que $p+q-pgcd(p,q) \leq |x|$, alors $pgcd(p,q)$ est aussi une période de x.*

Soit ζ un cycle couvrant solide de longueur ℓ avec le décalage d appartenant au graphe de de Bruijn pairé G. On doit montrer qu'il existe un cycle couvrant de longueur $pgcd(d,\ell)$ dans G. Pour cela, on considère le cycle ζ comme un mot z de longueur ℓ où chaque nœud correspond à une lettre de l'alphabet V. On considère également le mot $z^2 = z \cdot z$. Ce mot possède une longueur 2ℓ et une période ℓ. De plus, ζ est un cycle solide et H est une union de boucles. Ainsi, pour chaque $z^2[i]$, $i \leq |z^2| - d$, on a $z^2[i] = z^2[i+d]$. En conséquence, par le Lemme 9, z^2 a également la période $pgcd(d,\ell)$. Cela implique les faits suivants : premièrement, le chemin $C = (v_1, \ldots, v_{pgcd(d,\ell)})$ consistant dans les $pgcd(d,\ell)$ premiers nœuds dans ζ est un cycle comme $z^2[pgcd(d,\ell)+1] = z^2[1]$, deuxièmement, C est également un cycle couvrant de G. En effet, ζ est un cycle couvrant, et comme z possède la période $pgcd(d,\ell)$, il n'y a aucune paire de symboles adjacents dans z qui ne figure pas dans $z[1, pgcd(d,\ell)]$. Par conséquent, il n'y a aucun arc dans ζ qui n'apparaît pas dans C. Le cycle C est donc couvrant.

Inversement, nous supposons que $C = (v_1, \ldots, v_{pgcd(d,\ell)})$ est un cycle couvrant de longueur $pgcd(d,\ell)$. Alors, $C^{\ell/pgcd(d,\ell)}$, c'est-à-dire, la concaténation de $\ell/pgcd(d,\ell)$ fois de C, est un cycle couvrant de longueur ℓ. Dans ce cycle, le nœud de la position i est identique au nœud de la position $i+d$ car la longueur des cycles C concaténés est un diviseur de d. En conséquence, C est également un cycle solide avec le décalage d. □

Théorème 6. *Le* PROBLÈME DU CYCLE COUVRANT SOLIDE *peut être résolu en $O(8^{n\cdot\Delta} \cdot 2^n \cdot \mathrm{poly}(n+\log \ell))$ si le graphe de compatibilité H de G est une union de boucles.*

Démonstration. Par le Lemme 8, le problème est réduit à la recherche d'un cycle couvrant de longueur $\ell' \leq \ell$. Comme G possède n nœuds et au plus $n \cdot \Delta$ arcs, cela peut être fait en $O(8^{n\cdot\Delta} \cdot 2^n \cdot \mathrm{poly}(n+\log \ell))$ par le Théorème 1. □

6.8 Conclusion

Les problèmes liés à l'assemblage avec le graphe de de Bruijn pairé sont algorithmiquement beaucoup plus difficiles à traiter que sur le graphe de de Bruijn classique. Cela est dû à l'inclusion des informations des reads pairés dans la structure du graphe qui augmente de façon considérable sa complexité. Pour rendre ce graphe plus accessible nous avons proposé des algorithmes pour différents problèmes d'assemblage avec le graphe de de Bruijn pairé. Il serait souhaitable d'améliorer les algorithmes présentés. Toute amélioration substantielle devra éviter l'énumération de tous les chemins de longueur d dans G. En outre, il serait intéressant d'étendre l'algorithme pour le cas où H est une union disjointe de boucles, par exemple dans le cas où chaque sommet dans H a un degré sortant égal à un.

Conclusion

Dans ce mémoire, nous nous sommes intéressés aux problèmes soulevés par les répétitions dans l'assemblage de novo. L'émergence des techniques de séquençage de nouvelle génération a contraint les algorithmes dédiés à l'assemblage de novo à s'adapter à des données plus fragmentées et contenant plus d'erreurs. Malgré les progrès réalisés par ces algorithmes, l'assemblage de novo d'un génome reste un défi très important en raison, notamment, de la présence de régions répétées. Après avoir analysé les limites des approches existantes face aux répétitions, nous avons proposé des solutions algorithmiques pour améliorer l'assemblage de répétitions suivant deux directions. La première s'est concentrée sur la détection d'un type spécifique de répétitions, les répétitions en tandem. Pour la seconde, nous avons proposé des réponses aux problèmes d'assemblage avec le graphe de de Bruijn pairé, structure facilitant l'assemblage des répétitions et améliorant la qualité de l'assemblage global.

Détection des répétitions en tandem

Notre première contribution consiste en la réalisation d'algorithmes efficaces de détection d'un type de répétitions particulier, les répétitions en tandem (RT). Les répétitions de ce type ont un rôle important dans l'évolution des génomes, dans la régulation transcriptomique et dans diverses maladies génétiques [40]. Les assembleurs existants peuvent résoudre certaines RT, mais, dans le cas des génomes complexes, un nombre important de RT n'est pas assemblé. Pour palier à ce problème, nous avons proposé deux algorithmes, DExTaR et MixTaR. Ces deux algorithmes sont basés sur le graphe de de Bruijn en raison de son efficacité dans l'assemblage de novo. Le graphe de de Bruijn est basé sur la découpe des reads en fragments de longueur k appelés k-mers. Cela permet la représentation des répétitions en tandem exactes (RTE) dans ce type de graphe sous la forme d'un modèle topologique bien identifiable : des cycles satisfaisant des propriétés spécifiques.

Dans un premier temps, notre algorithme nommé DExTaR vise à améliorer l'assemblage des RTE suite à un assemblage de de Bruijn. DExTaR est implémenté comme une procédure post-contigs qui peut être adaptée à tous les assembleurs de de Bruijn classiques. Notre algorithme récupère le graphe de de Bruijn construit par un assembleur de de Bruijn et détecte de nouvelles RTE en analysant les parties qui n'ont pas été résolues par l'assemblage. Cela est réalisé en trois étapes principales. Tout d'abord, le graphe est parcouru pour rechercher des cycles. Ensuite, les cycles sont analysés pour détecter ceux qui vérifient les propriétés spécifiques aux RTE et ainsi construire les séquences de répétitions correspondantes. Enfin, une étape de validation de ces séquences utilise les reads qui ont servi pour la construction du graphe de de Bruijn. Pour chaque RTE validée, DExTaR renvoie son motif, le nombre de copies ainsi que les séquences précédant et suivant la répétition. Ces dernières apportent des informations sur l'emplacement de la RTE sur

le fragment d'ADN cible.

Pour les résultats expérimentaux, nous avons utilisé l'assembleur ABySS [111] en raison de sa construction classique du graphe de de Bruijn. Nous avons testé notre méthode avec des reads courts pairés, simulés à partir du premier chromosome de *C. elegans*. DExTaR améliore la détection des RTE en trouvant de nouvelles RTE avec des longueurs de motifs beaucoup plus importantes que celles des RTE assemblées par ABySS. La grand qualité des résultats est également démontrée par l'exploitation quasi-complète du potentiel des cycles sur lesquels nous recherchons des RTE.

Cependant, DExTaR dépend des résultats obtenus par l'assembleur utilisé. L'objectif principal des assembleurs est de construire de longues séquences de grande qualité. Pour cela, ils utilisent des heuristiques qui peuvent laisser des RTE non-résolues ou assembler des RTE de façon erronée. Ces choix impactent le potentiel de l'espace de recherche pour notre algorithme. L'analyse des résultats de DExTaR peut donc être étendue à d'autres assembleurs pour étudier la robustesse de DExTaR face au choix de l'assembleur.

De plus, DExTaR pourrait être adapté pour aider les assembleurs à créer de meilleurs contigs, et donc de meilleurs scaffolds. Notre algorithme fournit des informations concernant l'emplacement des RTE dans le graphe. En assemblant les RTE avec les contigs qui les précédent et les suivent, DExTaR peut réduire le nombre de contigs et augmenter leur longueur.

Dans le cas de génomes complexes, le graphe de de Bruijn, même assemblé, contient un nombre très important de cycles ce qui augmente de façon considérable la complexité de DExTaR. L'espace de recherche est donc fortement réduit par une limitation de la longueur des cycles détectés et analysés. Il serait souhaitable d'augmenter la longueur des cycles analysés en ciblant les parties du graphe significatives pour la recherche des RTE.

Une autre limitation provient du fait que DExTaR ne recherche que des RTE. L'algorithme pourrait être généralisé à d'autres types de répétitions comme les répétitions en tandem approximatives (RTA). Cela peut être réalisé en développant l'étape d'analyse de cycles pour considérer d'autres types de cycles.

Dans un second temps, nous avons proposé un algorithme appelé MixTaR qui utilise deux types de *reads* obtenus par les nouvelles méthodes de séquençage : les reads courts et les reads longs. MixTaR pallie à certaines limites de DExTaR en détectant des RTE et des RTA sans avoir besoin d'un assemblage global préalable.

La bonne qualité des reads courts et la grande longueur des reads longs sont exploitées par MixTaR à travers trois étapes principales. La première étape est basée sur DExTaR, mais, au lieu d'utiliser un graphe de de Bruijn déjà assemblé, nous construisons le graphe de de Bruijn correspondant à l'ensemble des reads courts. Ensuite, MixTaR parcourt le graphe pour détecter les cycles, puis analyse les cycles identifiés pour en déduire la liste des RTE potentielles. Ces RTE peuvent représenter des sous-séquences de RTA avec le même motif du fragment d'ADN cible. Dans ce cas, des copies approximatives des motifs des RTE sont situées à côté des RTE dans le fragment d'ADN cible. Les RT qui contiennent une RTE formant un cycle dans le graphe de de Bruijn sont appelées RT robustes. Notre algorithme MixTaR recherche d'abord les RTE et ensuite les RT robustes les contenant. Cela est réalisé à partir de la liste des motifs des RTE potentielles obtenues pendant cette étape. Lors de la deuxième étape, nous utilisons l'ensemble des reads longs pour valider les motifs détectés pendant la première étape. Un motif est validé si MixTaR identifie au moins deux copies approximatives de ce motif situées côte à côte dans au moins un read long. La troisième et dernière étape consiste à identifier la séquence

exacte des RT robustes contenant les motifs validés. Pour cela, nous utilisons à nouveau l'ensemble des reads courts pour des assemblages Greedy locaux.

MixTaR a été testé avec des données simulées et réelles à partir du premier chromosome de *C. elegans* ainsi qu'à partir du génome de *L. pneumophila*. Pour une analyse complète de sa robustesse face aux erreurs des reads, nous avons utilisé des reads courts et des reads longs avec différents taux d'erreurs. Les résultats obtenus démontrent la bonne précision et la grande sensibilité de notre méthode. Avec des taux de faux positifs faibles, même pour des reads très erronés, MixTaR est capable de détecter correctement un très grand nombre de RT avec des longueurs de motifs variant dans un vaste intervalle.

Néanmoins, l'algorithme et l'étude de ses résultats peuvent être développés. MixTaR utilise un outil de détection des RT dans des fragments d'ADN connus pour identifier la position des RT dans les contigs obtenus suite aux assemblages Greedy locaux. Pour nos tests, nous avons choisi l'un des plus connus, à savoir *mreps* [60]. Cependant, chaque outil qui détecte les RT dans des fragments d'ADN connus possède sa propre définition des RTA. Ainsi, les ensembles de RT détectées sont différents [71]. En conséquence, l'étude peut être étendue à d'autres outils de recherche de RT, comme par exemple *Tandem repeat finder* [7]. Cela est également valable pour l'assembleur Greedy utilisé.

En outre, en assemblant les séquences des RT, MixTaR récupère également les séquences précédant et suivant les RT. En conséquence, l'analyse de la qualité des résultats peut être étendue à ces régions adjacentes aux RT.

La définition d'une RT robuste pour MixTaR est limitée aux RT contenant une RTE interne du même motif et formant un cycle dans le graphe de de Bruijn. Mais cette définition peut être généralisée. Dans un graphe de de Bruijn, un cycle peut être formé par chaque répétition exacte d'un motif ayant une longueur minimale de k. Ainsi, la définition des RT robustes peut être étendue aux RT contenant une répétition exacte (celle-ci n'étant pas nécessairement en tandem) avec un motif de longueur minimale de k.

Assemblage basé sur le graphe de de Bruijn pairé

Une des nouvelles structures proposées dans la recherche de méthodes d'assemblage plus efficaces avec des données de séquençage de nouvelle génération est le graphe de de Bruijn pairé [82]. Le graphe de de Bruijn pairé inclut l'information des reads pairés. Grâce à une taille d'insert significative, les reads pairés sont capables de connecter des régions contenant de longues répétitions. L'utilisation de l'information des reads pairés pour la construction du graphe permet donc d'améliorer la qualité des résultats de l'assemblage ainsi que celle de la détection des répétitions. Cependant, le problème d'assemblage devient plus difficile lorsque l'on utilise ce type de graphe [57]. Par conséquent, notre travail est axé sur les problèmes algorithmiques déclenchés par l'assemblage de novo d'un génome avec un graphe de de Bruijn pairé.

Un assemblage correct dans le graphe de de Bruijn pairé est représenté par un chemin dit *couvrant*, c'est-à-dire comprenant tous les arcs du graphe. De plus, ce chemin couvrant reconstruit une paire de séquences d'ADN de sorte que la première séquence soit décalée par rapport à l'autre d'un nombre de caractères égal à la taille d'insert. Cette propriété supplémentaire est appelée contrainte de *solidité* (*soundness constraint*) et a été définie dans [57]. Cet article présente également une analyse de la complexité du problème de recherche de cycle couvrant dit *solide* (*sound*) dans le graphe de de Bruijn pairé. En particulier, dans le cas où k, la longueur des k-mers, et la taille de l'alphabet des reads pairés $|\Sigma|$ sont fixées, le problème de recherche du cycle solide couvrant dans le graphe

de de Bruijn pairé est indiqué comme étant polynomial sans pour autant que les auteurs ne proposent d'algorithme.

Nous avons développé un algorithme pour la recherche du cycle solide couvrant dans un graphe de de Bruijn pairé. Pour cela, nous avons établi une décomposition des cycles solides dans les graphes orientés et nous avons développé un algorithme pour calculer un cycle couvrant de longueur ℓ fixe. Le cycle solide couvrant peut être retrouvé par notre algorithme en un temps $f(n, d)\cdot$poly(log ℓ), où n représente le nombre de nœuds dans le graphe de de Buijn pairé et d la taille d'insert. Ainsi, nous évitons une complexité exponentielle en ℓ, ℓ ayant généralement une valeur beaucoup plus grande que d. Pour un cas particulier du graphe de de Buijn pairé, nous avons développé un algorithme qui retrouve un cycle solide couvrant en un temps $f(n)\cdot$poly(log ℓ + log d). Cela a été rendu réalisable en utilisant une nouvelle approche pour la modélisation des graphes de de Bruijn pairés appelée *graphe de compatibilité*. Nous avons également développé des algorithmes similaires pour des variantes du problème de recherche de cycles solides couvrants dans le graphe de de Bruijn pairé, telles que la recherche du plus court cycle solide couvrant et la recherche du cycle solide couvrant en considérant une taille d'insert approximative.

Pour rendre possible l'utilisation de ce graphe dans la pratique, la complexité des algorithmes présentés devra être encore réduite. En effet, les algorithmes proposés utilisent une étape d'énumération de tous les chemins de longueur d dans le graphe de de Bruijn pairé. La complexité de cette étape est de $O(n \cdot \Delta^{d-1})$, Δ étant le degré sortant maximum pour les nœuds du graphe de de Bruijn pairé. En supprimant cette étape, la complexité de nos algorithmes ne serait plus exponentielle en d.

Table des matières

Liste des tableaux

Table des figures

Liste des Algorithmes

Bibliographie

[1] Jung Ho Ahn. ccTSA : A Coverage-Centric Threaded Sequence Assembler. *PLOS ONE*, 7(6) :e39232, 2012. 26

[2] Wilhelm J. Ansorge. Next-generation DNA sequencing techniques. *New Biotechnology*, 25(4) :195–203, 2009. 15

[3] Pramila Nuwantha Ariyaratne and Wing-Kin Sung. PE-Assembler : de novo assembler using short paired-end reads. *Bioinformatics*, 27(2) :167–174, 2011. 21

[4] Kin Fai Au, Jason G. Underwood, Lawrence Lee, and Wing Hung Wong. Improving PacBio long read accuracy by short read alignment. *PLOS ONE*, 7(10) :e46679, 2012. 27, 81, 83, 84

[5] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, et al. SPAdes : a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5) :455–477, 2012. 27, 29

[6] Ali Bashir, Aaron A. Klammer, William P. Robins, Chen-Shan Chin, Dale Webster, Ellen Paxinos, David Hsu, Meredith Ashby, Susana Wang, Paul Peluso, et al. A hybrid approach for the automated finishing of bacterial genomes. *Nature Biotechnology*, 30(7) :701–707, 2012. 27

[7] Gary Benson. Tandem repeats finder : a program to analyze DNA sequences. *Nucleic Acids Research*, 27(2) :573–580, 1999. 38, 43, 104, 129

[8] Alain Bernot. *Analyse de génomes, transcriptomes et protéomes*. Dunod, 2001. 14

[9] Sebastian Böcker and Zsuzsanna Lipták. A fast and simple algorithm for the money changing problem. *Algorithmica*, 48(4) :413–432, 2007. 116, 119, 120

[10] Valentina Boeva, Mireille Regnier, Dmitri Papatsenko, and Vsevolod Makeev. Short fuzzy tandem repeats in genomic sequences, identification, and possible role in regulation of gene expression. *Bioinformatics*, 22(6) :676–684, 2006. 38

[11] Ma'ayan Bresler, Sara Sheehan, Andrew H. Chan, and Yun S. Song. Telescoper : de novo assembly of highly repetitive regions. *Bioinformatics*, 28(18) :i311–i317, 2012. 23

[12] Douglas W. Bryant, Weng-Keen Wong, and Todd C. Mockler. QSRA–A quality-value guided de novo short read assembler. *BMC Bioinformatics*, 10(1) :69, 2009. 21

[13] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. *Technical Report 124*, 1994. 39

[14] Jonathan Butler, Iain MacCallum, Michael Kleber, Ilya A. Shlyakhter, Matthew K. Belmonte, Eric S. Lander, Chad Nusbaum, and David B. Jaffe. ALLPATHS : de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5) :810–820, 2008. 88

[15] Matt J. Cahill, Claudio U. Köser, Nicholas E. Ross, and John A.C. Archer. Read length and repeat resolution : exploring prokaryote genomes using next-generation sequencing technologies. *PLOS ONE*, 5(7) :e11518, 2010. 67

[16] Mauricio O. Carneiro, Carsten Russ, Michael G. Ross, Stacey B. Gabriel, Chad Nusbaum, and Mark A DePristo. Pacific biosciences sequencing technology for genotyping and variation discovery in human data. *BMC Genomics*, 13(1) :375, 2012. 15, 81

[17] Sean B. Carroll, Jennifer K. Grenier, and Scott D. Weatherbee. *From DNA to diversity : molecular genetics and the evolution of animal design*. John Wiley & Sons, 2013. 13

[18] Adalberto T. Castelo, Wellington Martins, and Guang R. Gao. TROLL—tandem repeat occurrence locator. *Bioinformatics*, 18(4) :634–636, 2002. 37

[19] Mark J. Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR) : application and theory. *BMC Bioinformatics*, 13(1) :238, 2012. 15, 27, 81, 83, 84, 133

[20] Yu-Jung Chang, Chien-Chih Chen, Chuen-Liang Chen, and Jan-Ming Ho. A de novo next generation genomic sequence assembler based on string graph and MapReduce cloud computing framework. *BMC Genomics*, 13(Suppl 7) :S28, 2012. 22

[21] Jarrod A. Chapman, Isaac Ho, Sirisha Sunkara, Shujun Luo, Gary P. Schroth, and Daniel S. Rokhsar. Meraculous : de novo genome assembly with short paired-end reads. *PLOS ONE*, 6(8) :e23501, 2011. 26

[22] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. In *Proceedings of WABI 2012*, volume 7534 of *LNBI*, pages 236–248. Springer, 2012. 26

[23] David A. Coil, Liesbeth Vandersmissen, Christophe Ginevra, Sophie Jarraud, Elke Lammertyn, and Jozef Anné. Intragenic tandem repeat variation between Legionella pneumophila strains. *BMC Microbiology*, 8(1) :218, 2008. 43, 56, 97, 98, 99, 100, 101, 133

[24] Francis S. Collins, Michael Morgan, and Aristides Patrinos. The Human Genome Project : lessons from large-scale biology. *Science*, 300(5617) :286–290, 2003. 14

[25] Viraj Deshpande, Eric DK Fung, Son Pham, and Vineet Bafna. Cerulean : A hybrid assembly using high throughput short and long reads. In *Algorithms in Bioinformatics*, volume 8126 of *LNCS*, pages 349–363. Springer, 2013. 27

[26] Juliane C. Dohm, Claudio Lottaz, Tatiana Borodina, and Heinz Himmelbauer. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research*, 17(11) :1697–1706, 2007. 21

[27] Nevzat Onur Domaniç and Franco P. Preparata. A novel approach to the detection of genomic approximate tandem repeats in the levenshtein metric. *Journal of Computational Biology*, 14(7) :873–891, 2007. 38

[28] Andreas Döring, David Weese, Tobias Rausch, and Knut Reinert. SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, 9(1) :11, 2008. 85

[29] Erwan Drezen, Guillaume Rizk, Rayan Chikhi, Charles Deltel, Claire Lemaitre, Pierre Peterlongo, and Dominique Lavenier. Gatb : Genome assembly & analysis tool box. *Bioinformatics*, 30(20) :2959–2961, 2014. 85

[30] Dent Earl, Keith Bradnam, John St John, Aaron Darling, Dawei Lin, Joseph Fass, Hung On Ken Yu, Vince Buffalo, Daniel R. Zerbino, Mark Diekhans, et al. Assemblathon 1 : A competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12) :2224–2241, 2011. 26, 28, 43

[31] Jack Edmonds and Ellis L Johnson. Matching, Euler tours and the chinese postman. *Mathematical Programming*, 5(1) :88–124, 1973. 110

[32] Sara El-Metwally, Taher Hamza, Magdi Zakaria, and Mohamed Helmy. Next-Generation Sequence Assembly : Four Stages of Data Processing and Computational Challenges. *PLoS Comput. Biol.*, 9(12) :e1003345+, 2013. 22

[33] Hans Ellegren. Microsatellites : simple sequences with complex evolution. *Nature Reviews Genetics*, 5(6) :435–445, 2004. 34

[34] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000. 39

[35] Guillaume Fertin, Géraldine Jean, Andreea Radulescu, and Irena Rusu. DExTaR : Detection of exact tandem repeats based on the de Bruijn graph. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2014*, pages 90–93. IEEE, 2014. 61

[36] Guillaume Fertin, Géraldine Jean, Andreea Radulescu, and Irena Rusu. Hybrid de novo tandem repeat detection using short and long reads. *BMC Medical Genomics*, 8(Suppl 3) :S5, 2015. 77

[37] Nathan J. Fine and Herbert S Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1) :109–114, 1965. 125

[38] John W. Fondon and Harold R. Garner. Molecular origins of rapid and continuous morphological evolution. *Proceedings of the National Academy of Sciences USA*, 101(52) :18058–18063, 2004. 33

[39] Michael R. Garey and David S. Johnson. *Computers and Intractability : a guide to NP-completeness*. 1979. 22

[40] Yevgeniy Gelfand, Alfredo Rodriguez, and Gary Benson. TRDB—the tandem repeats database. *Nucleic Acids Research*, 35(suppl 1) :D80–D87, 2007. 127

[41] Rita Gemayel, Marcelo D. Vinces, Matthieu Legendre, and Kevin J. Verstrepen. Variable tandem repeats accelerate evolution of coding and regulatory sequences. *Annual review of genetics*, 44 :445–477, 2010. 34

[42] Greg Gibson and Spencer V Muse. *Précis de génomique*. De Boeck Supérieur, 2004. 11, 13

[43] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Joshua N. Burton, Bruce J. Walker, Ted Sharpe, Giles Hall, Terrance P. Shea, Sean Sykes, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences USA*, 108(4) :1513–1518, 2011. 26

[44] Dan Gusfield. *Algorithms on Strings, Trees and Sequences : Computer Science and Computational Biology*. Cambridge University Press, 1997. 35

[45] Thomas Hackl, Rainer Hedrich, Jörg Schultz, and Frank Förster. proovread : large-scale high-accuracy PacBio correction through iterative short read consensus. *Bioinformatics*, 30(21) :3004–3011, 2014. 27, 81, 83, 84

[46] David Hernandez, Patrice François, Laurent Farinelli, Magne Østerås, and Jacques Schrenzel. De novo bacterial genome sequencing : millions of very short reads assembled on a desktop computer. *Genome Research*, 18(5) :802–809, 2008. 22

[47] Mohammad S. Hossain, Navid Azimi, and Steven Skiena. Crystallizing short-read assemblies around seeds. *BMC Bioinformatics*, 10(Suppl 1) :S16, 2009. 23

[48] Ramana M. Idury and Michael S Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2) :291–306, 1995. 23

[49] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2) :226–232, 2012. 26

[50] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2) :226–232, 2012. 28

[51] Géraldine Jean, Andreea Radulescu, and Irena Rusu. The contig assembly problem and its algorithmic solutions. In Mourad Elloumi, editor, *Algorithms for Next-Generations Sequencing Data : Techniques Approaches and Applications*. Springer, 2016. (in press). 18

[52] William R. Jeck, Josephine A. Reinhardt, David A. Baltrus, Matthew T. Hickenbotham, Vincent Magrini, Elaine R. Mardis, Jeffery L. Dangl, and Corbin D. Jones. Extending assembly of short DNA sequences to handle error. *Bioinformatics*, 23(21) :2942–2944, 2007. 21

[53] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1) :77–84, 1975. 62, 80

[54] Julien Jorda and Andrey V Kajava. T-REKS : identification of Tandem REpeats in sequences with a K-meanS based algorithm. *Bioinformatics*, 25(20) :2632–2638, 2009. 38

[55] Jerzy Jurka, Vladimir V. Kapitonov, Oleksiy Kohany, and Michael V Jurka. Repetitive sequences in complex genomes : structure and evolution. *Annual Reviews of Genomics Human Genetics*, 8 :241–259, 2007. 6, 33

[56] Jerzy Jurka, Vladimir V. Kapitonov, A. Pavlicek, P. Klonowski, O. Kohany, and J. Walichiewicz. Repbase Update, a database of eukaryotic repetitive elements. *Cytogenetic and Genome Research*, 110(1-4) :462–467, 2005. 37

[57] Evgeny Kapun and Fedor Tsarev. On NP-Hardness of the Paired de Bruijn Sound Cycle Problem. In *Proceedings of WABI 2013*, volume 8126 of *LNBI*, pages 59–69. Springer, 2013. 29, 107, 110, 129

[58] John D. Kececioglu and Eugene W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13 :7–51, 1993. 22

[59] Dimitrios Kleftogiannis, Panos Kalnis, and Vladimir B. Bajic. Comparing Memory-Efficient Genome Assemblers on Stand-Alone and Cloud Infrastructures. *PLOS ONE*, 8(9) :e75505, 2013. 23, 26

[60] Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps : efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Research*, 31(13) :3672–3678, 2003. 38, 40, 41, 42, 43, 48, 86, 87, 104, 129, 137

[61] Roman Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 596–604. IEEE, 1999. 35

[62] Roman Kolpakov and Gregory Kucherov. Finding approximate repetitions under Hamming distance. *Theoretical Computer Science*, 303(1) :135–156, 2003. 37, 40, 41, 42, 137

[63] Christian Komusiewicz and Andreea Radulescu. On the Sound Covering Cycle Problem in Paired de Bruijn Graphs. In *Proceedings of FAW 2015*, volume 9130 of *LNCS*. Springer, 2015. 107

[64] Sergey Koren, Michael C. Schatz, Brian P. Walenz, Jeffrey Martin, Jason T. Howard, Ganeshkumar Ganapathy, Zhong Wang, David A. Rasko, W. Richard McCombie, Erich D. Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology*, 30(7) :693–700, 2012. 27, 81, 83

[65] Arun Krishnan and Francis Tang. Exhaustive whole-genome tandem repeats search. *Bioinformatics*, 20(16) :2702–2710, 2004. 38

[66] Sébastien Leclercq, Eric Rivals, and Philippe Jarne. Detecting microsatellites within genomes : significant variation among algorithms. *BMC Bioinformatics*, 8(1) :125, 2007. 37

[67] Richard M. Leggett, Ricardo H. Ramirez-Gonzalez, Walter Verweij, Cintia G. Kawashima, Zamin Iqbal, Jonathan DG Jones, Mario Caccamo, and Daniel MacLean. Identifying and classifying trait linked polymorphisms in non-reference species by walking coloured de Bruijn graphs. *PLOS ONE*, 8(3) :e60058, 2013. 28

[68] Claire Lemaitre, Liviu Ciortuz, and Pierre Peterlongo. Mapping-free and assembly-free discovery of inversion breakpoints from raw NGS reads. In *Algorithms for Computational Biology*, pages 119–130. Springer, 2014. 28

[69] Wai K. Leung, Jae J. Kim, Jong G. Kim, David Y. Graham, and Antonia R. Sepulveda. Microsatellite instability in gastric intestinal metaplasia in patients with and without gastric cancer. *The American Journal of Pathology*, 156(2) :537–543, 2000. 33

[70] Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2) :265–272, 2010. 26

[71] Kian Guan Lim, Chee Keong Kwoh, Li Yang Hsu, and Adrianto Wirawan. Review of tandem repeat search tools : a systematic approach to evaluating algorithmic performance. *Briefings in Bioinformatics*, 14(1) :67–81, 2013. 37, 104, 129

[72] Yong Lin, Jian Li, Hui Shen, Lei Zhang, Christopher J. Papasian, et al. Comparative studies of de novo assembly tools for next-generation sequencing technologies. *Bioinformatics*, 27(15) :2031–2037, 2011. 21

[73] Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems. *BioMed Research International*, 2012, 2012. 15, 133

[74] Yongchao Liu, Bertil Schmidt, and Douglas L. Maskell. Parallelized short read assembly of large genomes using de Bruijn graphs. *BMC Bioinformatics*, 12 :354, 2011. 26

[75] Monsieur Lothaire. *Algebraic combinatorics on words*, volume 90. Cambridge University Press, 2002. 34

[76] Iain MacCallum, Dariusz Przybylski, Sante Gnerre, Joshua Burton, Ilya Shlyakhter, Andreas Gnirke, Joel Malek, Kevin McKernan, Swati Ranade, Terrance P. Shea, Louise Williams, Sarah Young, Chad Nusbaum, and David B. Jaffe. ALLPATHS 2 : small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology*, 10, 2009. 26

[77] Udi Manber and Gene Myers. Suffix arrays : a new method for on-line string searches. *siam Journal on Computing*, 22(5) :935–948, 1993. 38

[78] Allan M. Maxam and Walter Gilbert. A new method for sequencing DNA. *Proceedings of the National Academy of Sciences USA*, 74(2) :560–564, 1977. 14

[79] Christoph Mayer, Florian Leese, and Ralph Tollrian. Genome-wide analysis of tandem repeats in Daphnia pulex-a comparative approach. *BMC Genomics*, 11(1) :277, 2010. 13, 33

[80] Kerensa E. McElroy, Fabio Luciani, and Torsten Thomas. GemSIM : general, error-model based simulator of next-generation sequencing data. *BMC Genomics*, 13(1) :74, 2012. 44, 70, 87

[81] Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *Proceedings of WABI 2007*, volume 4645 of *LNCS*, pages 289–301. Springer, 2007. 24

[82] Paul Medvedev, Son Pham, Mark Chaisson, Glenn Tesler, and Pavel Pevzner. Paired de Bruijn graphs : a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, 18(11) :1625–1634, 2011. 6, 29, 107, 129

[83] Angelika Merkel and Neil Gemmell. Detecting short tandem repeats from genome data : opening the software black box. *Briefings in Bioinformatics*, 9(5) :355–366, 2008. 37, 38, 43

[84] Angelika Merkel and Neil J. Gemmell. Detecting microsatellites in genome data : variance in definitions and bioinformatic approaches cause systematic bias. *Evolutionary bioinformatics online*, 4 :1, 2008. 37

[85] Jason R. Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6) :315–327, 2010. 18, 71, 82

[86] Kazuharu Misawa. RF : A method for filtering short reads with tandem repeats for genome mapping. *Genomics*, 102(1) :35–37, 2013. 35

[87] Michael Mitas. Trinucleotide repeats associated with human disease. *Nucleic acids research*, 25(12) :2245–2253, 1997. 33

[88] Suresh B. Mudunuri and Hampapathalu A. Nagarajaram. IMEx : imperfect microsatellite extractor. *Bioinformatics*, 23(10) :1181–1187, 2007. 38

[89] Francesca Nadalin, Francesco Vezzi, and Alberto Policriti. GapFiller : a de novo assembly approach to fill the gap within paired reads. *BMC Bioinformatics*, 13(Suppl 14) :S8, 2012. 21

[90] Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly : theory and applications to next generation sequencing. *Journal of Computational Biology*, 16(7) :897–908, 2009. 24

[91] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM : PacBio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1) :119–121, 2013. 83, 88

[92] Marco Pellegrini, M. Elena Renda, and Alessio Vecchio. TRStalker : an efficient heuristic for finding fuzzy tandem repeats. *Bioinformatics*, 26(12) :i358–i366, 2010. 38

[93] Hannu Peltola, Hans Söderlund, and Esko Ukkonen. SEQAID : a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Res.*, 12(1) :307–321, 1984. 21

[94] Yu Peng, Henry C.M. Leung, S.M. Yiu, and Francis Y.L. Chin. IDBA–a practical iterative de Bruijn graph de novo assembler. In *Research in Computational Molecular Biology*, pages 426–440. Springer, 2010. 26

[95] Pierre Peterlongo, Nicolas Schnel, Nadia Pisanti, Marie-France Sagot, and Vincent Lacroix. Identifying SNPs without a reference genome by comparing raw reads. In *String Processing and Information Retrieval*, pages 147–158. Springer, 2010. 28

[96] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences USA*, 98(17) :9748–9753, 2001. 23, 28, 68

[97] Rafal Pokrzywa and Andrzej Polanski. BWtrs : a tool for searching for tandem repeats in DNA sequences based on the Burrows–Wheeler transform. *Genomics*, 96(5) :316–321, 2010. 38, 39, 40, 70, 137

[98] Wayne Powell, Gordon C. Machray, and Jim Provan. Polymorphism revealed by simple sequence repeats. *Trends in Plant Science*, 1(7) :215–222, 1996. 13

[99] Andrey D. Prjibelski, Irina Vasilinetc, Anton Bankevich, Alexey Gurevich, Tatiana Krivosheeva, Sergey Nurk, Son Pham, Anton Korobeynikov, Alla Lapidus, and Pavel A. Pevzner. ExSPAnder : a universal repeat resolver for DNA fragment assembly. *Bioinformatics*, 30(12) :i293–i301, 2014. 29

[100] Michael A. Quail, Miriam Smith, Paul Coupland, Thomas D. Otto, Simon R. Harris, Thomas R. Connor, Anna Bertoni, Harold P. Swerdlow, and Yong Gu. A tale of three next generation sequencing platforms : comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics*, 13(1) :341, 2012. 15, 133

[101] Matt Ridley. *Genome*. Harper and Collins, 2000. 12

[102] Mohammed Sahli and Tetsuo Shibuya. Arapan-S : a fast and highly accurate whole-genome assembly software for viruses and small genomes. *BMC Research Notes*, 5(1) :243, 2012. 26

[103] Leena Salmela and Eric Rivals. LoRDEC : accurate and efficient long read error correction. *Bioinformatics*, page btu538, 2014. 27, 81, 83

[104] Steven L. Salzberg, Adam M. Phillippy, et al. GAGE : A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, 22(3) :557–567, 2012. 6, 22, 23, 28, 43, 44, 46, 68, 83, 88

[105] Frederick Sanger, Steven Nicklen, and Alan R Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences USA*, 74(12) :5463–5467, 1977. 14

[106] Michael C. Schatz, Arthur L. Delcher, and Steven L. Salzberg. Assembly of large genomes using second-generation sequencing. *Genome Research*, 20(9) :1165–1173, 2010. 22

[107] Bertil Schmidt, Ranjan Sinha, Bryan Beresford-Smith, and Simon J Puglisi. A fast hybrid short read fragment assembly algorithm. *Bioinformatics*, 25(17) :2279–2280, 2009. 23

[108] Prakash C. Sharma, Atul Grover, and Günter Kahl. Mining microsatellites in eukaryotic genomes. *Trends in Biotechnology*, 25(11) :490–498, 2007. 33, 37

[109] Jared T. Simpson and Richard Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12) :i367–i373, 2010. 22

[110] Jared T. Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3) :549–556, 2012. 22, 43

[111] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and İnanç Birol. ABySS : a parallel assembler for short read sequence data. *Genome Research*, 19(6) :1117–1123, 2009. 26, 43, 44, 68, 128

[112] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1) :195–197, 1981. 19

[113] Dina Sokol, Gary Benson, and Justin Tojeira. Tandem repeats over the edit distance. *Bioinformatics*, 23(2) :e30–e35, 2007. 38

[114] Daniel Sommer, Arthur Delcher, Steven Salzberg, and Mihai Pop. Minimus : a fast, lightweight genome assembler. *BMC Bioinformatics*, 8(1) :64, 2007. 22

[115] Jens Stoye and Dan Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science*, 270(1) :843–856, 2002. 34

[116] Subbaya Subramanian, Rakesh K. Mishra, and Lalji Singh. Genome-wide analysis of microsatellite repeats in humans : their abundance and density in specific genomic regions. *Genome Biology*, 4(2) :R13, 2003. 13, 33, 74

[117] Todd J. Treangen and Steven L. Salzberg. Repetitive DNA and next-generation sequencing : computational challenges and solutions. *Nature Reviews Genetics*, 13(1) :36–46, 2012. 6, 28, 43, 57, 59, 60, 76

[118] Raluca Uricaru, Guillaume Rizk, Vincent Lacroix, Elsa Quillery, Olivier Plantard, Rayan Chikhi, Claire Lemaitre, and Pierre Peterlongo. Reference-free detection of isolated SNPs. *Nucleic acids research*, page gku1187, 2014. 28

[119] John Craig Venter, Mark D. Adams, Eugene W. Myers, Peter W. Li, Richard J. Mural, Granger G. Sutton, Hamilton O. Smith, Mark Yandell, Cheryl A. Evans, Robert A. Holt, et al. The sequence of the human genome. *Science*, 291(5507) :1304–1351, 2001. 33

[120] Kevin J. Verstrepen, An Jansen, Fran Lewitter, and Gerald R Fink. Intragenic tandem repeats generate functional variability. *Nature Genetics*, 37(9) :986–990, 2005. 33, 74

[121] Paolo Visca, Silvia D'Arezzo, Françoise Ramisse, Yevgeniy Gelfand, Gary Benson, Gilles Vergnaud, Norman K. Fry, and Christine Pourcel. Investigation of the population structure of Legionella pneumophila by analysis of tandem repeat copy number and internal sequence variation. *Microbiology*, 157(9) :2582–2594, 2011. 43, 56, 97, 98, 99, 100, 101, 133

[122] Nikolay Vyahhi, Alex Pyshkin, Son Pham, and Pavel A. Pevzner. From de Bruijn graphs to rectangle graphs for genome assembly. In *Proceedings of WABI 2012*, LNBI, pages 249–261. 2012. 29

[123] René L. Warren, Granger G. Sutton, Steven J.M. Jones, and Robert A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4) :500–501, 2007. 21, 43, 85, 88

[124] James D. Watson, Francis H. C. Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356) :737–738, 1953. 11

[125] Joshua Wetzel, Carl Kingsford, and Mihai Pop. Assessing the benefits of using mate-pairs to resolve repeats in de novo short-read prokaryotic assemblies. *BMC Bioinformatics*, 12(1) :95, 2011. 28

[126] Ydo Wexler, Zohar Yakhini, Yechezkel Kashi, and Dan Geiger. Finding approximate tandem repeats in genomic sequences. *Journal of Computational Biology*, 12(7) :928–942, 2005. 38

[127] Adrianto Wirawan, Chee Keong Kwoh, Li Yang Hsu, and Tse Hsien Koh. INVERTER : integrated variable number tandem repeat finder. In *Computational Systems-Biology and Bioinformatics*, pages 151–164. Springer, 2010. 43

[128] Chengxi Ye, Chris Hill, Jue Ruan, and Zhanshan (Sam) Ma. DBG2OLC : Efficient assembly of large genomes using the compressed overlap graph. *arXiv preprint arXiv :1410.2801*, 2015. 27

[129] Chengxi Ye, Zhanshan S. Ma, Charles H. Cannon, Mihai Pop, and Douglas W. Yu. Exploiting sparseness in de novo genome assembly. *BMC Bioinformatics*, 13(Suppl 6) :S1, 2012. 26

[130] Daniel R. Zerbino and Ewan Birney. Velvet : algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5) :821–829, 2008. 26, 43

[131] Daniel R. Zerbino, Gayle K. McEwen, Elliott H. Margulies, and Ewan Birney. Pebble and Rock Band : Heuristic Resolution of Repeats and Scaffolding in the Velvet Short-Read *de Novo* Assembler. *PLOS ONE*, 4(12) :e8407, 2009. 28

[132] Wenyu Zhang, Jiajia Chen, Yang Yang, Yifei Tang, Jing Shang, and Bairong Shen. A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. *PLoS One*, 6(3) :e17915, 2011. 6, 21, 26, 28, 43, 57, 85

[133] Zhixin Zhao, Cheng Guo, Sreeskandarajan Sutharzan, Pei Li, Craig S. Echt, and Jie Zhang. Genome-Wide Analysis of Tandem Repeats in Plants and Green Algae. *G3 : Genes| Genomes| Genetics*, 4(1) :67–78, 2014. 13, 33

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Andreea RADULESCU

Assemblage de novo de répétitions à partir de données NGS

De novo repeat assembly from NGS data

Résumé

Le développement des méthodes de séquençage de nouvelle génération a permis la production de grandes quantités de données à moindre coût. Cependant, les fragments obtenus, appelés reads, possèdent des longueurs plus courtes et des taux d'erreurs plus élevés que ceux obtenus avec les premières méthodes de séquençage. Cela a créé de nouveaux défis pour l'assemblage de génomes. Même si de nombreux assembleurs sont publiés chaque année et que les algorithmes sont de plus en plus élaborés, la reconstruction d'un génome entier *de novo*, en l'absence de génome de référence, reste un problème difficile. Une des principales causes est la présence des répétitions dans les génomes.

Cette thèse décrit des algorithmes visant à améliorer l'assemblage de novo de répétitions. Nous présentons d'abord nos solutions axées sur les répétitions en tandem. L'algorithme appelé DExTaR a été conçu pour améliorer la détection de répétitions en tandem exactes suite à un assemblage de novo global basé sur l'approche de de Bruijn. Le second algorithme, appelé MixTaR, effectue seulement des assemblages locaux afin de détecter des répétitions en tandem exactes et approximatives. En utilisant deux types de reads, courts et longs, MixTaR ne requiert pas un assemblage global préalable. Nous proposons ensuite plusieurs algorithmes pour simplifier le problème d'assemblage basé sur une nouvelle structure de données, le graphe de de Bruijn pairé. Ce graphe inclut les informations des reads pairés dès le début du processus d'assemblage afin d'améliorer la détection de répétitions et la qualité de l'assemblage.

Mots clés

séquençage de nouvelle génération, assemblage de novo, graphe de de Bruijn, répétitions en tandem, cycle couvrant solide.

Abstract

The development of the next-generation sequencing methods has allowed the generation of vast amounts of data at a lower cost and time. However, the fragments obtained, called reads, have shorter lengths and higher error rates that the ones obtained with the first sequencing methods. This new type of data created new challenges in genome assembly. Even though many assembly software are published every year and algorithms are becoming more and more complex, reconstructing a whole genome *de novo*, in the absence of a reference genome, remains a difficult problem. One of the main causes is represented by the presence of repetitive regions in the genomes.

This thesis describes algorithms designed to improve the de novo assembly of repeats. We first present our solutions focused on tandem repeats. The algorithm called DExTaR aims at extending the work done by a de novo assembly in the detection of exact tandem repeats. Based on a de Bruijn graph constructed by an assembler, our approach assembles new exact tandem repeats by analysing the parts of the graph left unresolved. The second algorithm, called MixTaR, performs only local assemblies in order to detect exact and approximate tandem repeats. Using the two types of reads obtained by the new sequencing methods, short and long reads, MixTaR does not require a global de novo assembly. We then propose several algorithms for simplifying the assembly problem based on a new data structure, the paired de Bruijn graph. This graph uses the paired-end information from the beginning of the assembly process as a solution to a better repeat detection and higher quality results.

Key Words

next generation sequencing, de novo assembly, de Bruijn graph, tandem repeats, sound covering cycle.