Correction de données de séquençage de troisième génération

Pierre Morisse

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de l'université de Rouen Normandie

Correction de données de séquençage de troisième génération

Présentée et soutenue par
Pierre MORISSE

Thèse soutenue publiquement le 26 septembre 2019 devant le jury composé de		
M. Guillaume BLIN	PR, Université de Bordeaux	Rapporteur
Mme Christine GASPIN	DR, INRA Toulouse	Rapporteuse
M. Gregory KUCHEROV	DR, CNRS – Université Paris-Est Marne-La-Vallée	Examinateur
M. Thierry LECROQ	PR, Université de Rouen Normandie	Directeur de thèse
M. Arnaud LEFEBVRE	MCF, Université de Rouen Normandie	Co-encadrant de thèse
M. Pierre PETERLONGO	CR, INRIA Rennes	Examinateur
Mme Hélène TOUZET	DR, CNRS – Université de Lille	Examinatrice

Thèse dirigée par Thierry LECROQ, laboratoire LITIS

UNIVERSITÉ DE ROUEN NORMANDIE

MIIS

litis

Résumé

Les objectifs de cette thèse s'inscrivent dans la large problématique du traitement des données issues de séquenceurs à très haut débit, et plus particulièrement des *reads* longs, issus de séquenceurs de troisième génération. Les aspects abordés dans cette problématiques se concentrent principalement sur la correction des erreurs de séquençage, et sur l'impact de la correction sur la qualité des analyses sous-jacentes, plus particulièrement sur l'assemblage.

Dans un premier temps, l'un des objectifs de cette thèse est de permettre d'évaluer et de comparer la qualité de la correction fournie par les différentes méthodes de correction hybride (utilisant des *reads* courts en complément) et d'auto-correction (se basant uniquement sur l'information contenue dans les *reads* longs) de l'état de l'art. Une telle évaluation permet d'identifier aisément quelle méthode de correction est la mieux adaptée à un cas donné, notamment en fonction de la complexité du génome étudié, de la profondeur de séquençage, ou du taux d'erreurs des *reads*. De plus, les développeurs peuvent ainsi identifier les limitations des méthodes existantes, afin de guider leurs travaux et de proposer de nouvelles solutions visant à pallier ces limitations. Un nouvel outil d'évaluation, proposant de nombreuses métriques supplémentaires par rapport au seul outil disponible jusqu'alors, a ainsi été développé. Cet outil, combinant une approche par alignement multiple à une stratégie de segmentation, permet également une réduction considérable du temps nécessaire à l'évaluation. À l'aide de cet outil, un *benchmark* de l'ensemble des méthodes de correction disponibles est présenté, sur une large variété de jeux de données, de profondeur de séquençage, de taux d'erreurs et de complexité variable, de la bactérie *A. baylyi* à l'humain. Ce *benchmark* a notamment permis d'identifier deux importantes limitations des outils existants : les *reads* affichant des taux d'erreurs supérieurs à 30%, et les *reads* de longueur supérieure à 50 000 paires de bases.

Le deuxième objectif de cette thèse est alors la correction des *reads* extrêmement bruités. Pour cela, un outil de correction hybride, combinant différentes approches de l'état de l'art, a été développé afin de surmonter les limitations des méthodes existantes. En particulier, cet outil combine une stratégie d'alignement des *reads* courts sur les *reads* longs à l'utilisation d'un graphe de de Bruijn, ayant la particularité d'être d'ordre variable. Le graphe est ainsi utilisé afin de relier les *reads* alignés, et donc de corriger les régions non couvertes des *reads* longs. Cette méthode permet ainsi de corriger des *reads* affichant des taux d'erreurs atteignant jusqu'à 44%, tout en permettant un meilleur passage à l'échelle sur de larges génomes et une diminution du temps de traitement, par rapport aux méthodes de l'état de l'art les plus efficaces.

Enfin, le troisième objectif de cette thèse est la correction des *reads* extrêmement longs. Pour cela, un outil utilisant cette fois une approche par auto-correction a été développé, en combinant, de nouveau, différentes méthodologies de l'état de l'art. Plus précisément, une stratégie de calcul des chevauchements entre les *reads*, puis une double étape de correction, par alignement multiple puis par utilisation de graphes de de Bruijn locaux, sont utilisées ici. Afin de permettre à cette méthode de passer efficacement à l'échelle sur les *reads* extrêmement longs, la stratégie de segmentation mentionnée précédemment a été généralisée. Cette méthode d'auto-correction permet ainsi de corriger des *reads* atteignant jusqu'à 340 000 paires de bases, tout en permettant un excellent passage à l'échelle sur des génomes plus complexes, tels que celui de l'humain.

Mots-clés : Séquençage à haut débit, Correction d'erreurs, Assemblage, Graphe de de Bruijn, Alignement multiple

Abstract

The aims of this thesis are part of the vast problematic of high-throughput sequencing data analysis. More specifically, this thesis deals with long reads from third-generation sequencing technologies. The aspects tackled in this topic mainly focus on error correction, and on its impact on downstream analyses such a *de novo* assembly.

As a first step, one of the objectives of this thesis is to evaluate and compare the quality of the error correction provided by the state-of-the-art tools, whether they employ a hybrid (using complementary short reads) or a self-correction (relying only on the information contained in the long reads sequences) strategy. Such an evaluation allows to easily identify which method is best tailored for a given case, according to the genome complexity, the sequencing depth, or the error rate of the reads. Moreover, developpers can thus identify the limiting factors of the existing methods, in order to guide their work and propose new solutions allowing to overcome these limitations. A new evaluation tool, providing a wide variety of metrics, compared to the only tool previously available, was thus developped. This tool combines a multiple sequence alignment approach and a segmentation strategy, thus allowing to drastically reduce the evaluation runtime. With the help of this tool, we present a benchmark of all the state-of-the-art error correction methods, on various datasets from several organisms, spanning from the *A. baylyi* bacteria to the human. This benchmark allowed to spot two major limiting factors of the existing tools : the reads displaying error rates above 30%, and the reads reaching more than 50 000 base pairs.

The second objective of this thesis is thus the error correction of highly noisy long reads. To this aim, a hybrid error correction tool, combining different strategies from the state-of-the-art, was developped, in order to overcome the limiting factors of existing methods. More precisely, this tool combines a short reads alignment strategy to the use of a variable-order de Bruijn graph. This graph is used in order to link the aligned short reads, and thus correct the uncovered regions of the long reads. This method allows to process reads displaying error rates as high as 44%, and scales better to larger genomes, while allowing to reduce the runtime of the error correction, compared to the most efficient state-of-the-art tools.

Finally, the third objectif of this thesis is the error correction of extremely long reads. To this aim, a self-correction tool was developed, by combining, once again, different methologies from the state-of-the-art. More precisely, an overlapping strategy, and a two phases error correction process, using multiple sequence alignement and local de Bruijn graphs, are used. In order to allow this method to scale to extremely long reads, the aforementioned segmentation strategy was generalized. This self-correction methods allows to process reads reaching up to 340 000 base pairs, and manages to scale very well to complex organisms such as the human genome.

Keywords : High-throughput sequencing, Error correction, Assembly, De Bruijn graphs, Multiple Sequence Alignment

Remerciements

Tout d'abord, je tiens à remercier Thierry Lecroq et Arnaud Lefebvre de m'avoir accordé leur confiance, et de m'avoir laissé une grande autonomie pour mener à bien mes travaux. Leur soutien, leur disponibilité, et leurs nombreux conseils m'ont été extrêmement précieux tout au long de cette thèse. J'adresse également de sincères remerciements à Christophe Hancart. Merci de m'avoir transmis, dès la L1, ta passion pour l'algorithmique. Sans tes cours, je n'aurais probablement pas choisi cette voie.

Je tiens également à remercier les membres du jury, Guillaume Blin, Christine Gaspin, Gregory Kucherov, Pierre Peterlongo et Hélène Touzet d'avoir accepté d'évaluer ce travail.

Un grand merci également à l'équipe TIBS, qui m'a accueilli il y a maintenant plus de trois ans, pour la sympathie qu'ils m'ont témoignée. Élise, Laurent, Martine, Caroline, Nicolas, Hélène, Saïd et Lina : merci pour votre bonne humeur au quotidien.

J'adresse également de sincères remerciements aux départements STPI et ASI de l'INSA Rouen Normandie pour leur accueil lors de mes deux missions d'enseignement. Merci à Laurent, Romain, Julien, Nicolas M., JB, Mathieu, Nicolas D., Benoît, Géraldine, Pierrick et Alexandre d'avoir fait de ces deux premières années d'enseignement une expérience fort agréable.

Une pensée aussi au secrétariat du LITIS. Marion, Fabienne, Mathieu : merci d'avoir toujours assuré la préparation de mes missions, malgré les quelques (nombreux ?) couacs des agences de voyage, et mes demandes parfois un peu tardives.

Merci à toute l'équipe d'ELECTOR, Camille, Antoine, Lolita, et Pierre avec qui nous avons pu faire du bon travail, que nous allons, j'espère, poursuivre dans le futur.

Une autre pensée va également à tous mes co-bureau, passés et présents. Gabriel, Lysiane, Sylvestre, Maud et Vincent : merci de m'avoir tenu compagnie, plus ou moins longtemps, dans ce grand bureau parfois bien vide. Je vous souhaite un bon courage pour la suite, que vous ayez choisi la voie de la recherche ou non.

En parlant de voie alternative, merci aussi à Pierre, à Guillaume, à Raph, à Steeven, et aux autres copains de la Licence et du Master, qui n'ont pas choisi la recherche. Ces cinq années à vos côtés, remplies de bonne humeur et de rigolade, ont été fort agréables.

Des remerciements un peu plus particuliers à mes collègues doctorants ou déjà docteurs. Clément, merci à nos singes respectifs d'avoir entretenu nos moments

de procrastination lors de mes visites au Madrillet. Amazigh, merci de partager mon amour pour la nourriture, et merci d'avoir été un fidèle compagnon de tablée. Manon, merci d'avoir contrebalancé les bêtises des deux autres, et d'avoir permis des discussions plus sérieuses. Chloé, merci pour nos «courtes» pauses vape, nos longues discussions, et tes bons conseils séries.

Merci aussi à tous les collègues croisés en conférences, et qui ont rendu ces dernières aussi amusantes que scientifiquement intéressantes. Pierre, Quentin, Yannick, Wesley, Victor, et tous ceux que j'oublie probablement, bon courage pour la fin de vos thèses respectives, et j'espère que nous continuerons à nous croiser régulièrement.

Je tiens également à remercier Aika, Faïze, Adam, JR, Katja, Elisa, Miki, Marylise, Alexandre H., Pierre, Mél, Wes, Elisabeth, Marie, Aurélien, Hillary, Laurène, Jérôme, Bastien, Lucas, Alexandre B. (bon courage à toi aussi pour la suite de ta thèse au passage), et tous les autres amis extra-professionnels qui m'ont soutenu et m'ont permis de décompresser durant ces trois années, que ce soit autour d'une discussion, d'un (ou plusieurs) verre(s), ou pendant tout un festival.

Enfin, je tiens évidemment à remercier ma famille pour leur soutien et leurs encouragements, non seulement durant cette thèse, mais également durant tout le parcours qui m'a mené jusqu'ici.

Table des matières

Table des figures

Liste des tableaux

Nomenclature

ADN Acide DésoxyriboNucléique

ARN Acide RiboNucléique

BAM Binary Alignment/Map

bp base pairs

BWT Burrows-Wheeler Transform

C3G Correction de données de séquençage de 3ème Génération

CCD Charge-Coupled Device

CCS Circular Consensus Sequence

CLR Continuous Long Read

CPU Central Processing Unit

DAG Directed Acyclic Graph

DBG De Bruijn Graph

DDF Distance Difference Factor

ddNTP Didésoxyribonucléotides

HMM Hidden Markov Model

HTS High-Throughput Sequencing

ISFET Ion-Sensitive Field-Effect Transistor

LCP Longest Common Prefix

MEM Maximal Exact Match

MHAP MinHash Alignment Process

NGS Next Generation Sequencing

NSV Next Smaller Value

OEA One-End Anchors

ONT Oxford Nanopore Technologies

PacBio Pacific Biosciences

PAF Pairwise Alignement Format

PCR Polymerase Chain Reaction

pHMM profile Hidden Markov Model

PSV Previous Smaller Value

RAM Random Access Memory

SAM Sequence Alignment/Map

SMRT Single Molecule Real Time

SNV Single Nucleotide Variation

tf-idf term frequency, inverse document frequency

ZMW Zero-Mode Waveguides

Chapitre 1

Introduction

Sommaire

1.1 Préambule

Le traitement de données issues de séquenceurs à haut débit est un important champ d'étude du domaine de la bioinformatique. Ces données sont notamment utilisées afin de reconstruire de nouveaux génomes, pour lesquels des références ne sont pas disponibles, mais également afin de détecter des mutations, pouvant être des marqueurs de maladies génétiques. La récente apparition des technologies de séquençage de troisième génération complexifie cependant largement ces analyses, à cause des importants taux d'erreurs des données issues de ces technologies, pouvant atteindre 10 à 30%. Ces données de séquençage de troisième génération, grâce à leur longueur importante, sont extrêmement prometteuses pour la résolution de problèmes d'assemblage de génomes longs et complexes. Ainsi, le traitement et la correction de leurs erreurs constituent actuellement un défi majeur. L'objectif de cette thèse est donc d'étudier les méthodes existantes, permettant la correction de ces données, afin d'identifier leurs limites, et de proposer de nouveaux outils, permettant une correction plus efficace et un meilleur passage à l'échelle sur des génomes de grande envergure.

1.2 Contexte de travail

1.2.1 Le laboratoire LITIS et l'équipe TIBS

Le LITIS (Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes, EA 4108) est le laboratoire de recherche dans le domaine des sciences et technologies de l'information en Haute-Normandie, et est dirigé par le Pr. Thierry Paquet. Le laboratoire possède une démarche pluridisciplinaire, et associe ainsi des travaux autours de l'informatique, des mathématiques, du traitement de l'information, et de la reconnaissance de formes. Le laboratoire se divise en 7 équipes de

recherche : l'équipe Combinatoire et Algorithmes (C&A), l'équipe Apprentissage (App), l'équipe Traitement de l'Information en Biologie Santé (TIBS), l'équipe Quantification en Imagerie Fonctionnelle (QuantIF), l'équipe Systèmes de Transport Intelligent (STI), l'équipe Multi-agents, Interaction, Décision (MIND), et l'équipe Réseaux d'Interaction et Intelligence Collective (RI2C). De plus, le LITIS fait partie de la fédération NormaSTIC (CNRS 3638), en association avec le Groupe de REcherche en Informatique, Image, Automatique et Instrumentation (GREYC) de Caen, depuis janvier 2014.

Les travaux menés dans le cadre de cette thèse s'inscrivent dans les thématiques de recherche de l'équipe TIBS, dirigée par le Pr. Thierry Lecroq. Les thématiques générales des travaux de recherche de cette équipe sont l'indexation et la recherche d'information pertinentes au sein de données biologiques, et plus particulièrement de données issues de séquenceurs à haut débit. L'équipe TIBS combine ainsi des travaux de modélisation statistique, d'algorithmique du texte, d'ingénierie des connaissances, et de fouille de données, afin de de permettre le traitement de ces données biologiques.

1.2.2 Le traitement des données de séquençage

Depuis le milieu des années 2000, l'apparition des technologies de séquençage à haut débit force la biologie à faire face au traitement d'énormes quantités de données. Ces données sont formées par des millions, voire des milliards de séquences, appelées *reads*, et représentant des fragments du génome de l'organisme séquencé. Au vu des quantités de données générées par ces séquenceurs, le développement de structures de données et d'algorithmes permettant un traitement rapide et efficace est devenu un domaine de recherche majeur en bioinformatique.

Comme mentionné précédemment, ces *reads* sont notamment utilisés dans des problèmes d'assemblage, visant à reconstruire les génomes des organismes séquencés, et permettant ainsi l'accès à de nouveaux génomes de référence, mais également dans le cadre de la détection de mutations et de maladies génétiques. Cependant, ces *reads* sont imparfaits, et contiennent des erreurs de séquençage dont le taux et le profil varient en fonction de la technologie utilisée.

Les technologies dites de deuxième génération, apparues au milieu des années 2000, permettent ainsi de séquencer des *reads* de quelques centaines de paires de bases. Ces *reads* affichent des taux d'erreurs moyens de l'ordre 1%, la grande majorité de ces erreurs étant des substitutions.

Les technologies dites de troisième génération, apparues au début des années 2010, permettent quant à elles de séquencer des *reads* bien plus longs, pouvant atteindre quelques dizaines à quelques centaines de milliers de paires de bases. Malgré l'intérêt de ces *reads* longs pour la résolution de problèmes d'assemblage, ces derniers affichent également des taux d'erreurs bien plus élevés que les *reads* courts de deuxième génération, atteignant 10 à 30% en moyenne. De plus, le profil de ces erreurs est également plus complexe que celui des *reads* de deuxième génération, la grande majorité de ces erreurs étant des insertions et des délétions.

Ces erreurs de séquençage complexifiant grandement les analyses, particulièrement dans le cas des *reads* longs, des méthodes permettant d'identifier et de corriger ces erreurs sont alors nécessaires afin de pouvoir utiliser ces données de manière optimale. Ainsi, la problématique de la correction des erreurs de séquençage contenues dans les *reads* longs représente actuellement un important défi et un champ d'étude majeur dans le domaine de la bioinformatique.

1.3 Contexte de recherche

Les travaux présentés dans ce manuscrit portent donc sur le traitement des données issues de séquenceurs à haut débit, et plus particulièrement sur la correction des *reads* longs de troisième génération, et son impact sur les résultats d'analyses. Dans ce contexte, un projet a été mené, en collaboration avec différents laboratoires d'informatique et de biologie. Ce projet est décrit ci-dessous.

1.3.1 Projet C3G

Le projet C3G (Correction de données de séquençage de 3ème Génération) était un project de recherche financé dans le cadre du défi MASTODONS du CNRS. Ce défi permet le financement de projets interdisciplinaires ayant pour thématique le traitement de grandes quantités de données. Au vu des masses de données pouvant être générées par les technologies de séquençage à haut débit, le projet C3G s'inscrivait tout naturellement dans le cadre du défi MASTODONS. Entre 2017 et 2018, ce projet a donc permis la collaboration entre l'équipe TIBS du LITIS, l'équipe Méthodes et Algorithmes pour la Bioinformatique (MAB) du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), les équipes Interactions Plantes-Nématodes (IPN) et Santé des Plantes Interactions Biotiques Outils scientifiques Collectique (SPIBOC) de l'Institut Sophia Agrobiotech (ISA) de Sophia Antipolis, l'équipe GenScale de l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) de Rennes, l'équipe GOGEPP de l'Institut de Génétique, Environnement et Protection des Plantes (IGEPP) de Rennes, l'équipe Écologie et Évolution des Zoonoses (EEZ) du Centre de Biologie pour la Gestion des Populations (CBGP) de Montpellier, et l'équipe Ecological & Evolutionary Genomics (EEG) du laboratoire Evolutionary Biology & Ecology (EBG) de l'université libre de Bruxelles.

Le premier axe de ce projet était principalement destiné aux partenaires d'informatique, et consistait donc à développer des algorithmes et des méthodes bioinformatiques permettant le traitement et la correction des nombreuses erreurs contenues dans les données de séquençage de troisième génération. Ainsi, dans le cadre de cette thèse, une collaboration a été mise en place avec Lolita Lecompte, Antoine Limasset, Camille Marchet et Pierre Peterlongo, et a abouti au développement de deux des principaux résultats présentés dans ce manuscrit. L'un de ces résultats est également décrit dans la thèse de Camille Marchet [83].

Le second axe était alors destiné aux partenaires de biologie, et avait donc pour objectif d'évaluer les méthodes développées sur des données réelles, afin d'en valider l'efficacité.

1.4 Objectifs

Le premier objectif de cette thèse est le développement d'un outil permettant d'évaluer la qualité de la correction fournie par les méthodes existantes. En effet, de nombreuses méthodes permettant la correction des *reads* longs existent, et emploient aussi bien une approche hybride (*i.e.* utilisant des *reads* courts en complément des *reads* longs) qu'une approche d'auto-correction (*i.e.* corrigeant les *reads* longs en se basant uniquement sur les informations contenues dans leurs séquences). De plus, le comportement de ces méthodes peut grandement varier en fonction du génome étudié, du taux d'erreurs et de la longueur des *reads*, ainsi que de la profondeur de séquençage. La qualité de la correction ayant un impact important sur les analyses

sous-jacentes, et notamment sur la qualité des assemblages pouvant être générés, il est alors important de pouvoir aisément identifier quelle méthode est la mieux adaptée à un cas donné. Une telle méthode d'évaluation permet alors de réaliser des *benchmarks* comparatifs de grande envergure, incluant de nombreux outils de correction et des jeux de données variés. L'intérêt de réaliser de tels *benchmarks* est double. Pour les biologistes, ils permettent de déterminer aisément quelle méthode de correction utiliser en fonction des propriétés de leurs jeux de données. Pour les développeurs, ils permettent de comparer aisément leurs outils aux méthodes existantes, et peuvent alors les aider à adapter les paramètres de leurs outils afin d'en restreindre les limitations.

Le deuxième objectif de cette thèse est le développement d'un outil de correction hybride destiné au traitement des *reads* longs fortement bruités, affichant des taux d'erreurs supérieurs à 30%. En effet, sur de tels *reads*, les méthodes d'auto-correction passent difficilement à l'échelle, et les méthodes de correction hybride existantes sont limitées. La plupart de ces dernières fournissent en effet des résultats peu satisfaisants, ne réduisant pas suffisamment les taux d'erreurs des *reads*, ou affichant des temps d'exécution trop importants, limitant alors grandement leur application à de larges génomes. Permettre la correction de données affichant de tels taux d'erreurs est toutefois intéressant, non seulement en raison du challenge algorithmique que cela représente, mais également car le reséquençage est coûteux, et n'est donc pas accessible à l'intégralité des laboratoires.

Le troisième objectif de cette thèse est le développement d'un outil d'auto-correction, cette fois destiné au traitement des *reads* extrêmement longs. Les technologies de séquençage de *reads* longs évoluent en effet rapidement, et permettent ainsi de séquencer des *reads* de plus en plus longs. Cependant, les méthodes d'auto-correction actuelles ne passent pas à l'échelle sur de tels *reads*, et ne parviennent pas à produire la moindre correction. L'utilisation de *reads* atteignant des longueurs importantes facilite cependant la résolution des régions répétées, et permet ainsi de produire des assemblages de meilleure qualité. De ce fait, pouvoir corriger ces *reads* est primordial pour la résolution de problèmes d'assemblage de génomes longs et complexes, disposant de nombreuses répétitions.

1.5 Organisation du manuscrit

Ce manuscrit s'organise en 8 chapitres. Les Chapitres 1, 2 et 3 décrivent le contexte du travail mené dans le cadre de cette thèse, ses objectifs, ainsi que l'état de l'art et les définitions liées aux technologies de séquençage et à la correction de *reads* longs. Le Chapitre 4 présente une nouvelle méthode permettant d'évaluer en détail la qualité de la correction fournie par les différentes méthodes disponibles. Le Chapitre 5 introduit une nouvelle méthode de correction hybride, permettant de traiter des *reads* longs affichant de très importants taux d'erreurs, et ne pouvant être corrigés efficacement avec les méthodes existantes. Le Chapitre 6 propose une nouvelle méthode d'auto-correction, visant à traiter des *reads* extrêmement longs, sur lesquels aucune des méthodes d'auto-correction existantes ne passe à l'échelle. Le Chapitre 7 propose un *benchmark* complet des différentes méthodes de correction existantes, et des deux nouvelles méthodes développées dans le cadre de cette thèse, sur des jeux de données variés. Enfin, le Chapitre 8 propose une conclusion à cette thèse, et ouvre sur ses perspectives.

Chapitre 2

Le séquençage de l'ADN

Sommaire

2.1 Introduction

Ce chapitre présente le contexte global des travaux de cette thèse, en commençant par une brève description des principes biochimiques de l'ADN. Ces derniers ne sont pas présentés en détail, et des raccourcis sont volontairement pris afin de n'introduire que les informations importantes pour la suite de ce manuscrit. Un historique de l'évolution des technologies de séquençage, des premières découvertes

aux technologies les plus récentes, est également proposé. Les données de séquençage sont ensuite abordées d'un point de vue informatique, notamment en introduisant quelques structures de données classiques permettant leur traitement. Enfin, les principales problématiques liées à l'exploitation de ces données de séquençage, qui seront abordées au sein de ce manuscrit, sont présentées.

2.2 L'ADN d'un point de vue biochimique

L'acide désoxyribonucléique, ou ADN, est une molécule permettant de stocker l'information biologique nécessaire au développement, au fonctionnement, et à la reproduction de la grande majorité des êtres vivants. L'ADN se compose de quatre acides nucléiques, appelés *nucléotides* ou *bases* : l'adénine (A), la cytosine (C), la thymine (T) et la guanine (G). Ces nucléotides ont la propriété de former des paires : A s'appareille avec T, tandis que C s'appareille avec G. On dit alors que A et T et que C et G sont *complémentaires*. Une séquence composée d'un ensemble de nucléotides est alors appelée un *brin* d'ADN. Les brins d'ADN sont orientés, chacune de leurs extrémités se terminant par un groupe chimique différent. L'une de ces extrémités, appelée *extrémité 5'*, se termine avec un groupe phosphate, tandis que l'autre extrémité, appelée *extrémité 3'*, se termine avec un groupe hydroxyle. Les acides nucléiques sont ainsi synthétisés du sens 5' vers 3'. Les brins d'ADN peuvent également former des paires, en s'attachant à l'aide de liens hydrogène.

Les deux brins d'une paire ont alors la même longueur, les différents nucléotides de chaque brin formant des *paires de bases* (notées *bp* pour *base pairs*), et chaque brin étant orienté dans la direction opposée à l'autre. Chaque brin est alors appelé le *complémentaire inverse* de l'autre brin. Par exemple, le brin TTGCAT représente le complémentaire inverse du brin ATGCAA. Une molécule d'ADN se compose alors de deux brins complémentaires, formant ainsi sa structure en double hélice, illustrée dans la partie droite de la Figure 2.1. Cette structure a été découverte en 1953 par James Dewey Watson et Francis Harry Compton Crick [117], en se basant sur des travaux de Rosalind Franklin et de Raymond Gosling. Cette découverte a valu à Watson et à Crick d'obtenir le prix Nobel de physiologie ou médecine, en 1962.

L'ADN est ainsi présent dans trois principaux types d'organismes : les eucaryotes, possédant des cellules avec un noyau bordé par une membrane, ainsi que les bactéries et les archées, étant tous les deux des procaryotes, dont les cellules ne possèdent pas de noyau. Chez les eucaryotes, l'ADN est alors stocké et séparé du reste de la cellule, au sein du noyau, ce qui n'est pas le cas chez les procaryotes.

Certaines régions de l'ADN permettent de coder les *gènes*, c'est-à-dire les instructions nécessaires aux différents processus fonctionnels des cellules, tandis que d'autres régions permettent de réguler l'activité de ces gènes. L'ADN compose ainsi une ou plusieurs structures, appelées *chromosomes*, au sein de chaque organisme. L'ensemble du matériel génétique contenu dans les chromosomes d'un organisme est alors appelé son *génome*.

Les organismes *haploïdes* ne portent qu'une seule copie de chaque chromosome, tandis que les organismes *polyploïdes* portent plusieurs copies de chaque chromosome. L'humain est par exemple un organisme *diploïde*, et porte une paire de chacun de ses 23 chromosomes. La taille des génomes n'est cependant pas corrélée avec la complexité apparente des organismes. Divers exemples de tailles de génomes sont illustrés Table 2.1.

De plus, le contenu des génomes est structuré, et la distribution des bases qu'ils contiennent n'est donc donc pas aléatoire. Plusieurs phénomènes biologiques

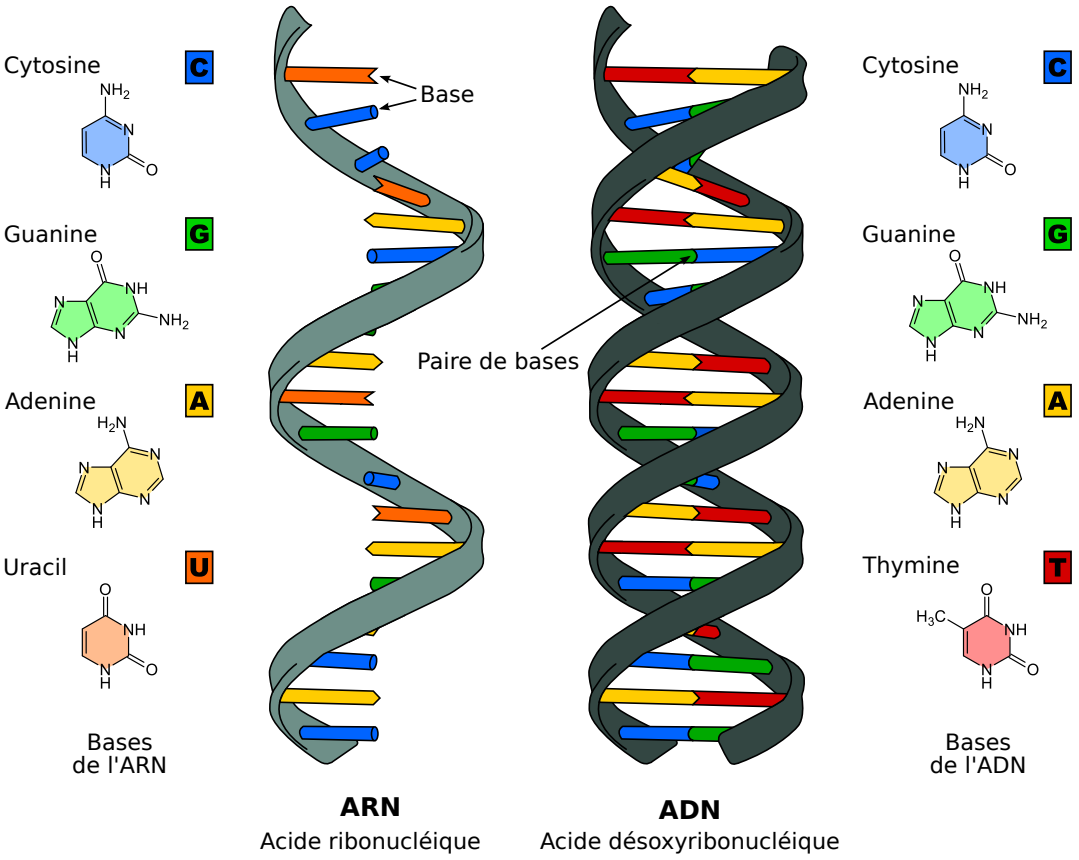

FIGURE 2.1 – **Composition et structure d'une molécule d'ADN et d'une molécule d'ARN.** Figure adaptée de [120].

Génome	Organisme	Taille
Acinetobacter baylyi	Bactérie	3,6 Mbp
Escherichia coli	Bactérie	4,6 Mbp
Saccharomyces cerevisiae	Levure	13 Mbp
Caenorhabditis elegans	Nématode	100 Mbp
Drosophila melanogaster	Drosophile	180 Mbp
Homo sapiens	Humain	3 300 Mbp
Ambystoma mexicanum	Axolotl	32 000 Mbp
Polychaos dubium	Amibe	670 000 Mbp

TABLE 2.1 – **Différents exemples de tailles de génomes.** Les génomes de *A. baylyi*, de *E. coli*, de *S. cerevisiae*, de *C. elegans*, de *D. melanogaster* et de *H. sapiens* seront utilisés pour mener des expériences au sein de ce manuscrit.

mènent en effet à la présence de *répétitions* au sein des génomes. Ces répétitions sont des séquences dupliquées, ayant plusieurs occurrences au sein du génome d'un organisme. Ces séquences peuvent cependant légèrement diverger, chacune des copies des répétitions pouvant être sujette à diverses mutations.

L'acide ribonucléique, ou ARN , est une seconde molécule présente au sein des êtres vivants. L'ARN se compose également d'acides nucléiques, et dispose de propriétés extrêmement similaires à celles de l'ADN. La thymine est cependant remplacée par l'uracile (U) au sein de l'ARN, qui n'est, de plus, généralement formé que d'un seul brin. La structure d'une molécule d'ARN est illustrée dans la partie gauche de la Figure 2.1. Un des rôles principaux de l'ARN est la diffusion de l'information portée par l'ADN aux cellules. En particulier, une catégorie de molécules d'ARN, appelées les *ARN messagers*, transporte les messages génétiques de l'ADN au reste de la cellule. Ces ARN messagers sont produits à partir des régions de l'ADN correspondant à des gènes : c'est le mécanisme de *transcription*. Ces ARN messagers sont ensuite décodés au sein de la cellule, afin de produire des *protéines* : c'est le mécanisme de *traduction*.

Contrairement à l'ADN et à l'ARN, les protéines sont composées d'acides aminées de 20 types différents. Le *code génétique* représente alors l'ensemble des règles permettant la traduction des nucléotides en acides aminés. Dans l'ARN messager, un triplet de trois nucléotides, appelé *codon*, permet de coder un acide aminé ou un *codon stop*. Un tel codon représente la fin d'une chaîne d'acides aminés. Les codons étant définis à partir de triplets de nucléotides, $4^3 = 64$ différents codons existent ainsi pour la définition de seulement 20 acides aminés et d'un codon stop. Certains acides aminés correspondent donc à plusieurs codons. Les protéines résultant de cette étape de traduction permettent alors diverses fonctions telles que le transport de molécules, les catalyses chimiques, ou la *réplication* de l'ADN. Ce dernier mécanisme permet d'obtenir deux molécules d'ADN identiques à partir d'une unique molécule d'ADN.

2.3 Technologies de séquençage

Le séquençage de l'ADN a été inventé en 1977, par deux équipes de recherche indépendantes. L'une d'entre elles était menée par Frederick Sanger, à l'université de Cambridge [101], et l'autre, par Allan Maxam et Walter Gilbert, à l'université de Harvard [85]. Bien qu'ayant eu lieu la même année, ces deux travaux de recherche sont totalement indépendants, et ont permis à Sanger et Gilbert de se voir tous deux attribuer le prix Nobel de chimie en 1980. En effet, cette découverte fut une avancée majeure dans le monde de la biologie, permettant d'étudier la composition ADN de toute vie biologique sur Terre, et ouvrant ainsi la porte à l'étude du code génétique des êtres vivants.

Le séquençage de l'ADN repose sur divers principes biochimiques, et vise à retranscrire l'information relative au génome, contenue dans un échantillon d'ADN, sous forme de chaînes de caractères, sur l'alphabet ADN de 4 lettres, $\Sigma = \{A, C, G, T\}$. Une cinquième lettre, N, peut toutefois être présente au sein des chaînes caractères produites. Cette lettre dénote alors une ambiguïté rencontrée lors du séquençage, et peut représenter n'importe laquelle des 4 bases. Ces chaînes de caractères, appelées *reads*, représentent des fragments de la séquence d'ADN de l'individu ou de l'espèce séquencé. De plus, ces *reads* sont imparfaits, et contiennent des erreurs de séquençage dont les types (substitution, insertion, ou délétion) et les taux (de 0,001 à 30%) varient en fonction des technologies utilisées. En effet, au vu

des enjeux du séquençage de l'ADN, les recherches à ce sujet se sont multipliées depuis 1977, et de nombreuses technologies de séquençage ont ainsi été développées.

Les technologies de Sanger et de Maxam-Gilbert se sont cependant longtemps imposées comme les technologies de séquençage les plus utilisées par les biologistes. En particulier, la technologie Sanger, au vu de sa haute efficacité et de sa faible radioactivité, a été plus largement adoptée que la technologie Maxam-Gilbert. Ces découvertes ont cependant grandement inspiré les chercheurs à développer de nouvelles technologies de séquençage, plus rapides, plus efficaces, et moins coûteuses. De ce fait, le séquençage de l'ADN a grandement évolué depuis son introduction, en 1977. Un historique des technologies de séquençage est ainsi présenté ici, des méthodes de Sanger et Maxam-Gilbert, dites de première génération, aux méthodes les plus récentes, dites de troisième génération. Les principales caractéristiques de ces technologies et plateformes de séquençage sont résumées Table 2.2. Bien que la littérature à ce sujet soit extrêmement riche ([106, 50, 34, 107, 37] par exemple), les aspects biochimiques du séquençage ne seront que brièvement abordés ici, puisque sortant du cadre de cette thèse.

2.3.1 Première génération

2.3.1.1 Sanger

La technologie Sanger repose sur une méthode de séquençage par synthèse. Elle utilise un des deux brins de l'échantillon d'ADN à séquencer comme modèle. Le séquençage est alors réalisé à l'aide de nucléotides chimiquement modifiés, appelés didésoxyribonucléotides (ddNTP). Ces ddNTP sont au nombre de 4, chacun associé à un nucléotide : ddATP, ddCTP, ddGTP et ddTTP. Une fois incorporés dans le brin d'ADN, à l'aide d'une ADN polymérase, ils permettent de stopper son élongation, au niveau des nucléotides associés au didésoxyribonucléotide incorporé. Par exemple, en incorporant un ddATP, l'élongation sera stoppée au niveau des nucléotides A. Ainsi, pour que le séquençage soit complet, il est nécessaire de réaliser cette réaction quatre fois en parallèle, avec les quatre différents didésoxyribonucléotides.

Chacune de ces réactions produit alors un ensemble de fragments d'ADN de tailles différentes, ces tailles étant fonction de l'endroit où a été incorporé le ddNTP. Ces fragments sont ensuite séparés selon leurs tailles, par électrophorèse, sur une plaque de gel de polyacrylamide. Une ligne indépendante de cette plaque de gel est utilisée pour chaque réaction. Ainsi, les positions des différents nucléotides dans la séquence peuvent être repérées par une bande, indiquant une incorporation de ddNTP, sur la ligne correspondante de la plaque de gel. Cette visualisation des nucléotides sous forme de bandes est réalisée à l'aide d'un système d'imagerie tel que les rayons X ou la lumière ultra-violette, et est rendue possible grâce à un traceur radioactif, préalablement incorporé dans l'ADN séquencé. Dans les versions ultérieures de cette technologie, ce traceur a pu être remplacé par un traceur fluorescent, afin de se débarrasser des inconvénients de la radioactivité. Le processus de séquençage Sanger est illustré Figure 2.2.

Le séquençage Sanger permet ainsi de produire des *reads* de longueur moyenne de 500 à 600 paires de bases, affichant un très faible taux d'erreurs, de l'ordre de 0,001%. De plus, la technologie Sanger, ayant été largement adoptée par les biologistes, a été utilisée pour séquencer et publier une première version du génome humain [31].

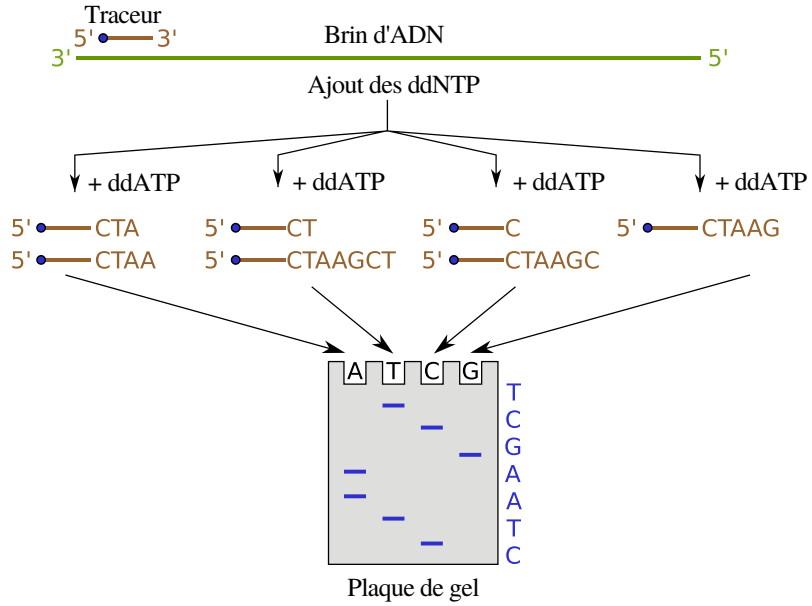

FIGURE 2.2 – **Séquençage Sanger.** Figure adaptée de [121].

2.3.1.2 Maxam-Gilbert

La technologie Maxam-Gilbert repose sur une méthode de séquençage par dégradation chimique. Elle utilise les deux brins de l'échantillon d'ADN à séquencer, qu'elle marque à l'aide d'un traceur radioactif, à l'extrémité 5'. Les deux bins de chaque fragment d'ADN sont ensuite séparés, et les réactivités différentes des quatre nucléotides sont alors utilisées afin de réaliser des coupures. Quatre types de réactions ont ainsi été mises au point, une pour les C, une pour les G, une pour les A et les G, et une pour les C et les T. Ces réactions sont effectuées en parallèle, sur une fraction de chaque brin d'ADN. Les concentrations des produits chimiques utilisés pour provoquer ces réactions sont contrôlées, de sorte qu'une seule modification ne soit apportée sur chaque molécule d'ADN.

Ces réactions produisent ainsi un ensemble de fragments de tailles différentes, en fonction de l'endroit où a été réalisée la coupure. Ces fragments sont ensuite séparés selon leurs tailles, par électrophorèse, sur une plaque de gel de polyacrylamide, comme pour la méthode de Sanger. De la même façon, les positions des différents nucléotides dans la séquence sont repérées par une bande sur la plaque de gel, à l'aide d'un système d'imagerie. Le processus de séquençage Maxam-Gilbert est illustré Figure 2.3.

Tout comme le séquençage Sanger, le séquençage Maxam-Gilbert permet également de produire des *reads* de longueur moyenne de 500 à 600 paires de bases, affichant un faible taux d'erreurs, de l'ordre de 0,001%.

2.3.2 Deuxième génération

Bien que les technologies de première génération, et particulièrement celle de Sanger, aient dominé le marché du séquençage durant près de trois décennies, le coût élevé et le temps nécessaire au séquençage demeuraient un facteur bloquant majeur. De ce fait, à partir de 2005, une nouvelle génération de séquenceurs a émergé, afin de surpasser ces limitations. Ces séquenceurs, dits de deuxième génération, réduisent

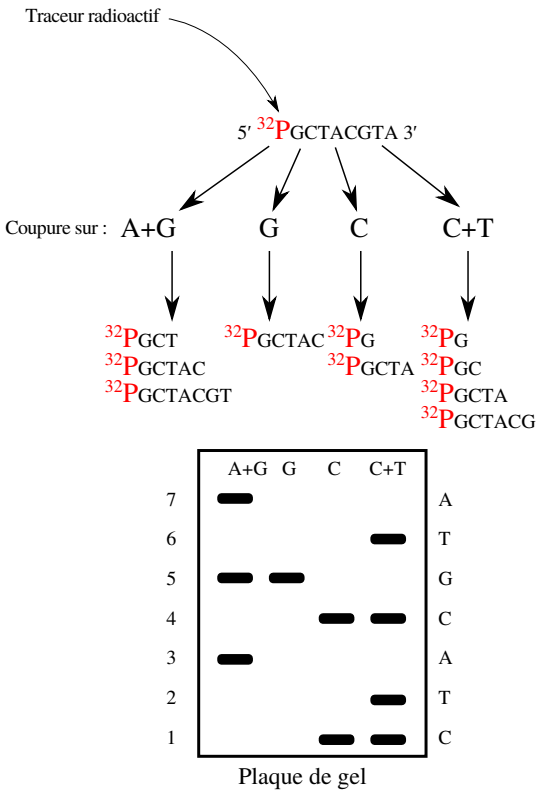

FIGURE 2.3 – **Séquençage Maxam-Gilbert.** Figure adaptée de [122].

grandement le coût du séquençage, et génèrent des millions de *reads* courts, en parallèle, permettant ainsi un important gain de temps par rapport aux séquenceurs de première génération. Cette génération de technologies de séquençage est généralement qualifiée de *NGS*, pour *Next Generation Sequencing*, ou de séquençage à haut débit (ou *HTS*, pour *High-Throughput Sequencing*). De plus, la sortie du séquençage peut ici être directement détectée, sans utiliser d'électrophorèse et de système d'imagerie. Les quatre principaux acteurs de ces technologies de deuxième génération sont présentés ici.

2.3.2.1 Roche / 454

La technologie Roche / 454 est apparue en 2005, et repose sur une méthode de pyroséquençage, qui est une approche de séquençage par synthèse. L'échantillon d'ADN à séquencer est fragmenté aléatoirement, puis des adaptateurs sont attachés aux extrémités des fragments d'ADN obtenus. Ces fragments d'ADN sont ensuite fixés à des billes, comportant des amorces complémentaires aux adaptateurs, permettant de retenir les fragments, et de s'assurer qu'une bille ne soit associée qu'à un seul fragment. Chaque bille est ensuite isolée, et le fragment d'ADN associé est amplifié par PCR (*Polymerase Chain Reaction*) en émulsion. Cette amplification permet de produire plusieurs millions de copies de chaque fragment d'ADN, à la surface des billes. Ces billes sont ensuite transférées sur une plaque PTP (*Pico Titer Plate*) contenant des milliers de puits, de diamètre suffisant pour ne récupérer qu'une seule bille. Les réactions de pyroséquençage ont alors lieu, en parallèle, dans chacun de ces puits.

Les 4 nucléotides sont ensuite ajoutés de manière séquentielle, et dans un ordre précis, sur la plaque PTP. Lors de son écoulement le long de la plaque, un nucléotide peut alors être incorporé aux fragments d'ADN à la surface des billes contenues dans les puits, à l'aide d'une ADN polymérase. Lors de cette incorporation, un pyrophosphate inorganique est libéré. Grâce à la présence de luciférase au sein des puits, ce pyrophosphate permet de générer un signal lumineux, d'intensité proportionnelle au nombre de nucléotides incorporés. Ainsi, l'incorporation d'un homopolymer (*i.e.* une région où la même base est répétée plusieurs fois) par l'écoulement d'un seul nucléotide produit un signal d'intensité plus importante que l'incorporation d'un unique nucléotide. La séquence d'un fragment d'ADN peut alors être déduite, en détectant la suite de signaux lumineux émise par l'incorporation des nucléotides, à l'aide d'un capteur CCD (*Charge-Couple Device*). Le processus de séquençage Roche / 454 est illustré Figure 2.4.

Le séquençage Roche / 454 permet ainsi de produire des *reads* relativement longs par rapport aux autres technologies de seconde génération, pouvant atteindre jusqu'à 700 paires de bases. Ces *reads* affichent un taux d'erreurs de 1% en moyenne, la majorité de ces erreurs étant des insertions et des délétions, en particulier dans les homopolymers. Ces erreurs s'expliquent par le fait que la taille des homopolymers est déterminée par l'intensité du signal lumineux émis par la réaction de pyroséquençage, causant ainsi une sur ou une sous-estimation du nombre de nucléotides incorporés, en cas de signal de trop forte ou de trop faible intensité.

2.3.2.2 Illumina / Solexa

La technologie Solexa et apparue en 2006. Elle a, par la suite, été renommée lors de son rachat par la société Illumina, en 2007. Elle repose sur une méthode de séquençage par synthèse. Pour cela, l'échantillon d'ADN à séquencer est tout d'abord fragmenté aléatoirement, et des adaptateurs sont attachés à chaque extrémité de ces fragments. Ces fragments sont alors, à leur tour, attachés à des *flow cells*, à l'aide de leurs adaptateurs. Ces *flow cells* contiennent en effet des oligonucléotides, capables de retenir les fragments d'ADN disposant d'un adaptateur complémentaire. Une fois les fragments attachés aux *flow cells*, ils sont amplifiés par PCR *bridge*, afin de créer plusieurs milliers de copies identiques de chaque fragment. Un ensemble de fragments provenant du même fragment original est alors appelé un *cluster*.

Une fois l'amplification terminée, et les *clusters* produits, des amorces, des ADN polymérases et des nucléotides modifiés sont ajoutés aux *flow cells*. Les amorces s'attachent alors sur les fragments d'ADN, qui peuvent être étendus par incorporation de nouveaux nucléotides, à l'aide de l'ADN polymérase. Les nucléotides ajoutés pour incorporation sont modifiés de telle sorte que chaque type de nucléotide soit identifié par un marqueur fluorescent unique. De plus, un terminateur réversible est également ajouté à chaque nucléotide, afin de garantir que l'ADN polymérase ne puisse incorporer qu'un seul nucléotide à la fois. À chaque incorporation, la base ayant été ajoutée est déterminée en fonction de la longueur d'onde de son marqueur fluorescent, détectée à l'aide d'un capteur CCD. Les nucléotides non incorporés, ainsi que le terminateur du nucléotide ayant été incorporé, sont ensuite retirés du *flow cell*. Le processus peut ainsi se poursuivre, avec une nouvelle incorporation de nucléotide, jusqu'à ce que le fragment d'ADN soit totalement séquencé. Le processus de séquençage Illumina est illustré Figure 2.5.

Le séquençage Illumina permet ainsi de produire des *reads* de longueur moyenne de 100 à 150 paires de bases, affichant un taux d'erreurs de 1% en moyenne. Des *reads* pouvant atteindre jusqu'à 300 paires de bases peuvent cependant être produits par

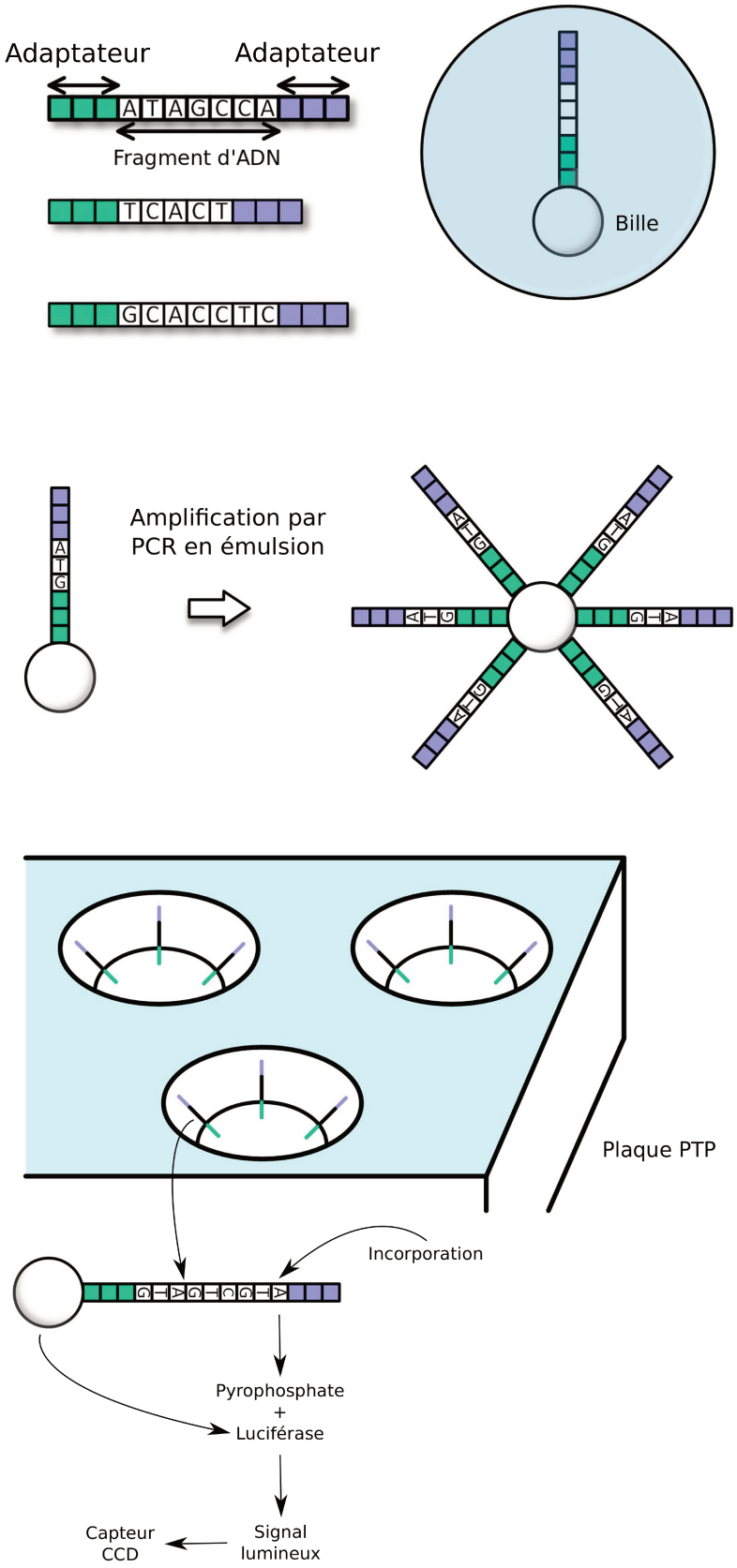

FIGURE 2.4 – **Séquençage Roche / 454.** Figure adaptée de [40].

certaines plateformes de séquençage. Les erreurs de séquençage de ces *reads* sont principalement des substitutions, dues à une mauvaise identification du nucléotide incorporé, et sont principalement situées aux extrémités 3' des *reads*. De plus, il est à noter qu'Illumina s'est imposée comme la technologie de séquençage de deuxième génération la plus utilisée.

2.3.2.3 ABI / SOLiD

La technologie ABI / SOLiD est apparue en 2007, et repose sur une méthode de séquençage par ligation. Pour cela, l'échantillon d'ADN à séquencer est tout d'abord fragmenté aléatoirement, puis les fragments d'ADN sont fixés à des billes à l'aide d'adaptateurs et d'amorces, et clonés par PCR en émulsion, comme pour la technologie Roche / 454. Ces billes sont ensuite déposées sur une lame de verre, sur laquelle elles s'attachent, grâce à des liaisons covalentes. La réaction de séquençage par ligation peut alors avoir lieu. Pour cela, des 8-mers (*i.e.* des fragments d'ADN de taille 8) spécifiques, disposant d'un marqueur fluorescent sur l'extrémité 5', sont utilisés. Ces 8-mers ont également une structure précise : les deux premières bases de l'extrémité 3' sont complémentaires aux nucléotides en cours de séquençage, les trois bases suivantes sont dégénérées, et peuvent s'appareiller avec n'importe quels nucléotides, et enfin, les deux dernières bases sont également dégénérées, mais sont retirées, avec le marqueur fluorescent, lors de la réaction.

L'étape de séquençage est composée de plusieurs passes, chacune de ces passes étant elle-même composée de plusieurs cycles. Au début de chaque passe, une amorce, complémentaire à celle utilisée pour fixer le fragment d'ADN à la bille, est ajoutée. Lors de chaque cycle, les 8-mers précédemment décrits sont ajoutés et ligaturés en fonction des deux premières bases de leur extrémité 3'. Les 8-mers n'ayant pas été liés au fragment d'ADN en cours de séquençage sont retirés de la lame de verre, et le signal fluorescent émis par le 8-mer ayant été lié est mesuré et enregistré. Au début de la passe suivante, une nouvelle amorce est ajoutée, et les cycles sont répétés comme lors de la première passe. Ces amorces sont choisies et ajoutées de telle sorte qu'elles se chevauchent, et que chaque base soit lue deux fois lors du séquençage. Les données obtenues par cette technologie de séquençage sont représentées par des couleurs, chacune d'entre elles décrivant deux bases. Ces données peuvent ensuite être traduites, afin de déterminer la séquence du fragment d'ADN séquencé. Le processus de séquençage ABI / SOLiD est illustré Figure 2.6.

Le séquençage ABI / SOLiD permet ainsi de produire des *reads* de longueur comprise entre 50 et 75, et affichant un taux d'erreurs très faible, de l'ordre de 0,1%, dû au fait que chaque base soit lue deux fois lors de séquençage. Ces erreurs sont principalement des substitutions, due à une mauvaise identifications des bases, causée par le bruit généré pendant les cycles de ligation.

2.3.2.4 Ion Torrent

La technologie Ion Torrent est apparue en 2010, et est basée sur la détection de l'ion hydrogène libéré lors de l'incorporation de nucléotides. Comme pour les technologies Roche / 454 et ABI / SOLiD, l'échantillon d'ADN à séquencer est fragmenté aléatoirement, et chaque fragment est attaché à une bille à l'aide d'adaptateurs et d'amorces, puis amplifié par PCR en émulsion. Une puce contenant un ensemble de puits, chacun d'entre eux contenant une bille, est ensuite utilisée.

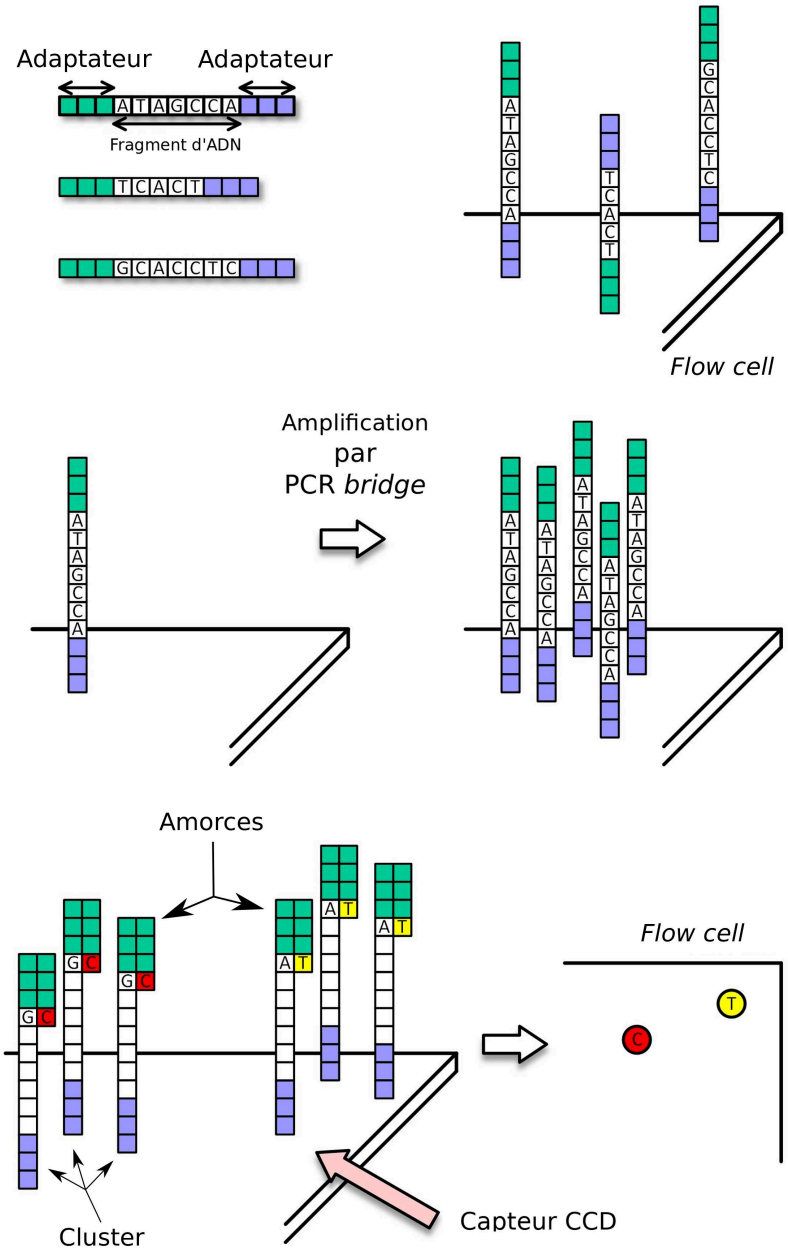

FIGURE 2.5 – **Séquençage Illumina.** Figure adaptée de [40].

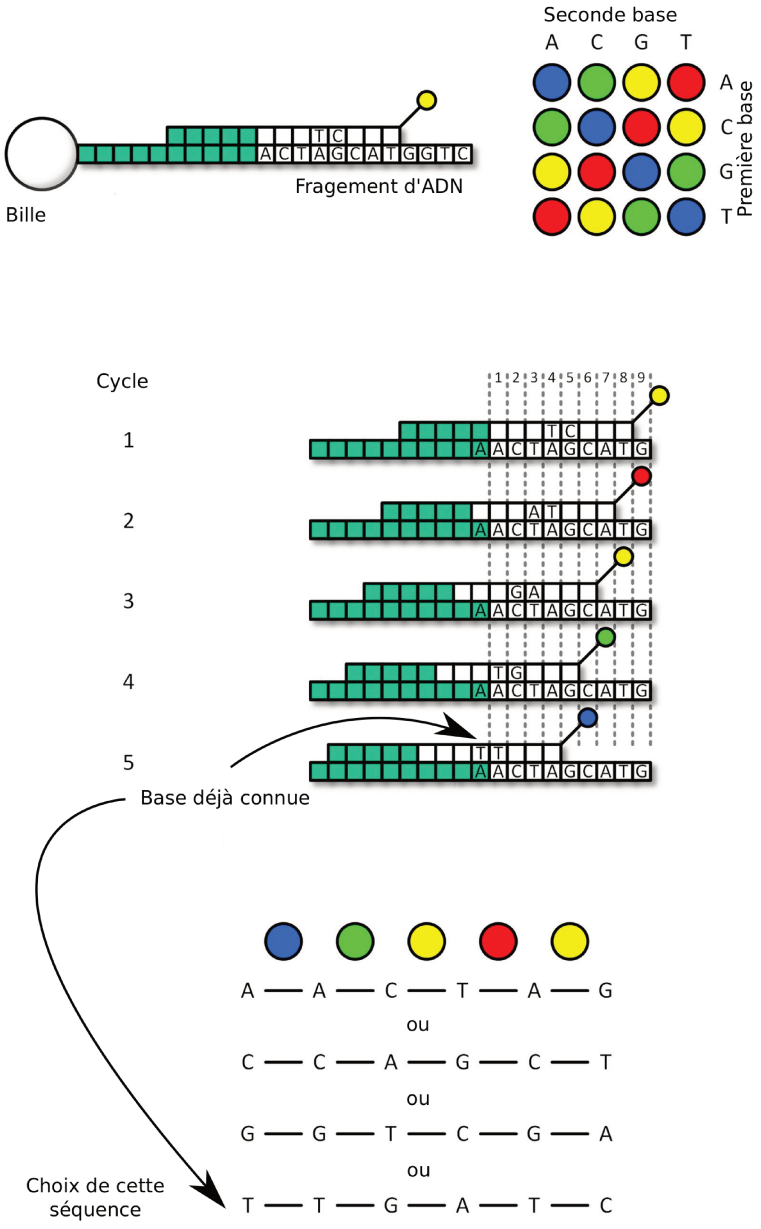

FIGURE 2.6 – **Séquençage ABI / SOLiD.** Figure adaptée de [40].

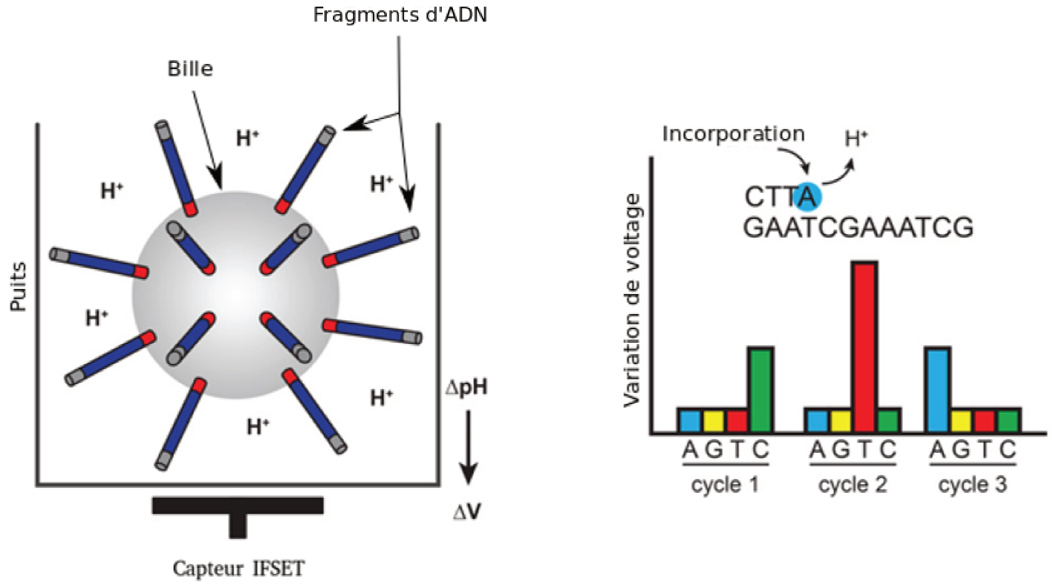

FIGURE 2.7 – **Séquençage Ion Torrent.** Figure adaptée de [93].

Les 4 nucléotides sont alors ajoutés séquentiellement, afin de permettre leur incorporation aux fragments d'ADN à la surface des billes. Cette incorporation, réalisée à l'aide d'une ADN polymérase, libère alors un ion hydrogène qui change le pH de la solution contenue dans le puits. Ce changement est détecté par un capteur ISFET (*Ion-Sensitive Field-Effect Transistor*) situé au fond du puits, puis converti en un signal électrique. Dans le cas où un homopolymer est incorporé à partir d'un unique nucléotide, le nombre d'ions hydrogènes libérés est proportionnel à la longueur de l'homopolymer, et l'intensité du signal électrique est alors plus importante. La série de signaux électriques ainsi émise par le capteur est transmise à un ordinateur, et traduite en une séquence d'ADN, sans besoin de conversion intermédiaire du signal. Le processus de séquençage Ion Torrent est illustré Figure 2.7.

Le séquençage Ion Torrent permet ainsi de produire des *reads* de longueur comprise entre 100 et 400, et affichant un taux d'erreurs de 1% en moyenne. Comme pour le séquençage Roche / 454, ces erreurs sont principalement des insertions et des délétions, en particulier dans les homopolymers, dû à la difficulté à correctement interpréter l'intensité du signal électrique produit.

2.3.2.5 *Reads* pairés

Ces séquenceurs de deuxième génération offrent également la possibilité de séquencer des *reads* pairés, dits *paired-end*. De tels *reads* proviennent chacun d'une des deux extrémités du fragment d'ADN séquencé. Pour chacun de ces *reads*, l'information concernant l'autre *read* lui étant appareillé, et la distance séparant les deux *reads* sur le fragment d'ADN séquencé sont alors connues. Le *read* pairé associé à un *read* court donné est généralement appelé *mate*. Ces informations sont notamment utiles lors de l'alignement de ces *reads*, afin de résoudre des réarrangements structuraux tels que des insertions, des délétions, ou des inversions, mais également lors de l'assemblage, en particulier dans les régions répétées.

2.3.3 Troisième génération

Bien qu'elles aient permis le séquençage de l'ADN à grande échelle, les technologies de séquençage de deuxième génération présentent quelques désavantages. Tout d'abord, toutes ces technologies reposent sur une étape longue et coûteuse d'amplification par PCR. De plus, la longueur des *reads* produits par les séquenceurs de deuxième génération est un facteur bloquant majeur dans l'analyse et l'assemblage de génomes complexes, plus particulièrement à cause de la difficulté de ces *reads* à résoudre les régions répétées. Ainsi, à partir de 2011, une nouvelle génération de séquenceurs, visant à résoudre ces problèmes, s'est développée.

Ces séquenceurs, dits de troisième génération, ne nécessitent pas d'étape d'amplification par PCR, et permettent ainsi un séquençage plus aisé, plus rapide, et moins coûteux que les séquenceurs de deuxième génération. De plus, ces séquenceurs de troisième génération permettent également de produire des *reads* bien plus longs, atteignant des longueurs moyennes de plusieurs milliers à plusieurs centaines de milliers de paires de bases. Un des principaux intérêts de ces *reads* longs est de contourner les difficultés rencontrées lors de l'utilisation de *reads* courts, de deuxième génération, dans des problèmes d'assemblage de génomes longs et complexes. En particulier, ces *reads*, grâce à leur importante longueur, permettent de mieux couvrir les régions répétées, et donc de les résoudre plus aisément. Cependant, ces *reads* sont également bien plus bruités que les *reads* de deuxième génération, et atteignent des taux d'erreurs compris entre 10 et 30%. De plus, alors que les *reads* de deuxième génération comportent une majorité d'erreurs de substitutions, les erreurs d'insertions et de délétions sont bien plus fréquentes dans les *reads* de troisième génération. Par ailleurs, ces séquenceurs de troisième génération sont également plus susceptibles de produire des *reads chimériques*, c'est-à-dire des *reads* formés de séquences qui ne sont pas contiguës au sein du génome de référence.

Les deux acteurs majeurs de ces technologies de troisième génération, Pacific Biosciences et Oxford Nanopore Techologies, sont présentées ici. D'autres technologies, telles que Illumina True-Seq (anciennement Moleculo) et 10x Genomics existent également, et peuvent être considérées comme des technologies de troisième génération. Cependant, elles reposent sur des méthodologies extrêmement différentes, et sont bien moins répandues que les technologies Pacific Biosciences et Oxford Nanopore Technologies. Par conséquent, ces technologies ne seront pas présentées. De plus, il est intéressant de noter que, contrairement aux technologies de deuxième génération, dominées par Illumina, aucune technologie de troisième génération ne se démarque à l'heure actuelle.

2.3.3.1 Pacific Biosciences

La technologie Pacfic Biosciences (PacBio) est apparue en 2011, et repose sur une approche de séquençage simple molécule en temps réel (SMRT, pour *Single Molecule Real Time*). Le processus repose sur un séquençage par synthèse similaire à l'approche utilisée par Illumina, mais plutôt que d'exécuter des cycles d'amplifications, le signal émis lors de l'incorporation d'un nucléotide est détecté en temps réel. Pour cela, des cellules contenant des *zero-mode waveguides* (ZMW), qui sont des puits de quelques dizaines de nanomètres de diamètre, sont utilisées. Au fond de ces ZMW, se situe une ADN polymérase, un seul fragment d'ADN, et un capteur permettant d'enregistrer les signaux lumineux. Durant la réaction de séquençage, des nucléotides identifiés à l'aide de marqueurs fluorescents sont incorporés à l'aide de l'ADN

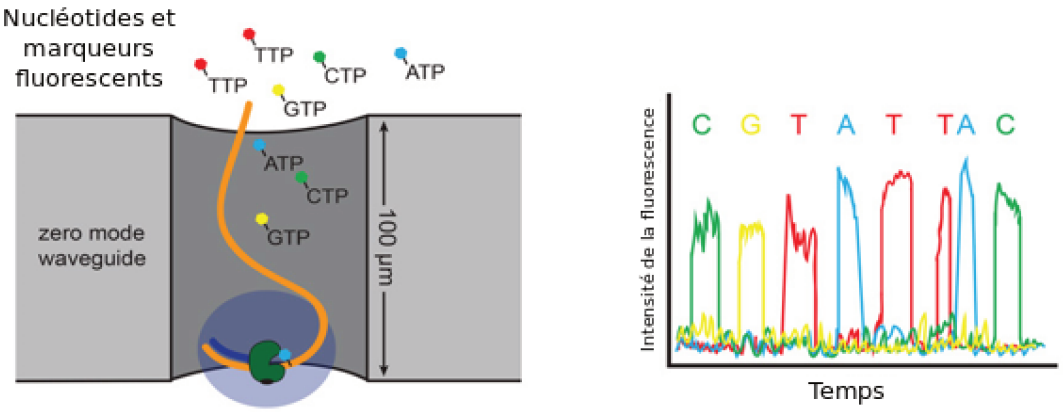

FIGURE 2.8 – **Séquençage PacBio.** Figure adaptée de [93].

polymérase, et un signal lumineux est émis à chaque incorporation. De plus, le marqueur fluorescent est détaché des nucléotides lors de l'incorporation, laissant ainsi le fragment d'ADN non modifié. De par la taille des ZMW, seul le signal lumineux émis le plus au fond du puits peut être détecté, permettant ainsi d'observer en temps réel les nucléotides incorporés par l'ADN polymérase, et donc de déterminer la séquence du fragment d'ADN. Le processus de séquençage PacBio est illustré Figure 2.8.

De plus, une stratégie permettant d'améliorer la précision du séquençage a été développée. Pour cela, des adaptateurs en épingle à cheveux sont ajoutés à chaque extrémité des deux brins d'un fragment d'ADN, afin de les relier, et de permettre à l'ADN polymérase de boucler autour du fragment. De cette façon, l'ADN polymérase peut traverser et séquencer plusieurs fois le fragment. Chaque séquence ainsi déterminée est appelée un *subread*. À partir de ces *subreads*, une séquence consensus de meilleure qualité peut alors être extraite. Cette séquence est appelée un *read CCS* (*Circular Consensus Sequence*). Les *reads* produits sans consensus sont, quant à eux, appelés des *reads CLR* (*Continuous Long Read*). La longueur et la qualité des *reads* CCS dépendent alors du nombre de boucles réalisées par l'ADN polymérase, et donc du nombre de *subreads* utilisés pour déterminer la séquence consensus. Ainsi, plus un *read* CCS sera précis, et plus il sera court.

Le séquençage PacBio permet ainsi de produire des *reads* atteignant en moyenne des longueurs de plusieurs milliers à quelques dizaines de milliers de paires de bases. Cependant, ces *reads* affichent un très haut taux d'erreurs, atteignant 10 à 15% en moyenne. Ces erreurs sont principalement des insertions et des délétions, avec une plus large proportion d'insertions, et sont distribuées aléatoirement sur toute la longueur des *reads*.

2.3.3.2 Oxford Nanopore Technologies

La technologie Oxford Nanopore Technologies (ONT) est apparue en 2014 en accès anticipé, et a été commercialisée en 2015. Cette technologie repose sur une approche de séquençage se basant sur des nanopores. Ces pores sont constitués de protéines, permettant de faire passer le fragment d'ADN. Lorsque le fragment d'ADN traverse le pore, le courant ionique de ce dernier varie. Chaque type de nucléotide provoque une variation différente du courant ionique dans le pore, et peut donc être

détecté. La variation du courant est ainsi enregistrée progressivement, et est ensuite interprétée afin de déterminer la séquence du fragment d'ADN.

De la même façon que pour les *reads* CCS proposés par PacBio, un adaptateur en épingle à cheveux peut être utilisé afin de relier les deux brins d'un fragment d'ADN. Dans ce cas, chaque brin passe alors dans le pore l'un après l'autre. Chaque séquence peut ensuite être détectée, puisque séparée de l'autre par l'adaptateur. Ces séquences peuvent alors être séparées, et former des *reads 1D*, ou utilisées afin de produire une séquence consensus de meilleure qualité, appelée *read 2D*. Cependant, cette méthodologie de *reads* 2D n'est plus maintenue depuis 2018, et est vouée à être remplacée par les *reads* $1D^2$, développés en 2017. Dans ce protocole, des protéines liées aux deux brins d'ADN leur permettent de passer le pore l'un après l'autre, mais les deux brins ne sont plus physiquement liés. Cette approche a été développée afin de pallier le fait que, dans certains cas, avec la technologie de *reads* 2D, l'adaptateur en épingle à cheveux pouvait produire une structure secondaire, ralentissant le passage du brin d'ADN à travers le nanopore, et impactant négativement les résultats. Cependant, dans l'approche $1D^2$, il est possible que les deux brins d'ADN ne passent pas à travers le pore l'un après l'autre, et qu'une séquence consensus ne puisse alors pas être obtenue. Le processus de séquençage ONT est illustré Figure 2.9.

ONT propose ainsi diverses plateformes de séquençage, dont la plateforme MinION, mesurant une dizaine de centimètres, et pouvant fonctionner à l'aide d'une simple connexion USB 3.0 à un ordinateur. Cette plateforme ouvre notamment la porte à un séquençage sur le terrain bien plus aisé, et peu coûteux, étant disponible au prix de $1 000. Une autre plateforme en cours de développement, le SmidgION, vise a réduire encore davantage la taille du séquenceur pour rendre le séquençage sur le terrain encore plus accessible, en permettant cette fois de le connecter à un simple smartphone.

Le séquençage ONT permet ainsi de produire des *reads* atteignant, à l'heure actuelle, des longueurs moyennes comprises entre quelques dizaines de milliers et 200 000 paires de bases. De plus, la longueur des *reads* séquencés ne dépend que de la préparation de la librairie, et non de la plateforme de séquençage utilisée. De ce fait, des *reads* atteignant des longueurs de plus d'un million de paires de bases, appelés *ultra-long reads*, ont pu être séquencés récemment [46]. Tout comme les *reads* PacBio, les *reads* ONT affichent également un haut taux d'erreurs, variant de 12 à 30% en fonction des versions des chimies utilisées pour le séquençage. Tout comme pour les *reads* PacBio, ces erreurs sont principalement des insertions et des délétions, mais les délétions sont cette fois présentes en plus large proportion. De plus, contrairement aux *reads* PacBio, les *reads* ONT ont tendance à contenir des erreurs systématiques dans les homopolymers. Ces erreurs sont dues à la difficulté de déterminer la longueur précise des homopolymers avec ce protocole, lorsque celle-ci atteint 5 à 6 bases.

2.3.4 Remarques et notations

La notion de *profondeur*, ou de *couverture*, d'une expérience de séquençage est définie comme la redondance induite par le séquençage. Elle représente le nombre de fois où chaque base du génome est couverte par un *read*. Une expérience de séquençage peut être vue comme produisant une distribution uniforme de *reads*, tout le long du génome séquencé. La couverture peut ainsi être définie comme le nombre total de nucléotides produits par l'expérience de séquençage, divisé par la taille du génome. Ainsi, par exemple, pour un génome d'un million de paires de bases, une expérience de séquençage produisant 1 million de *reads* de longueur 200 permettra

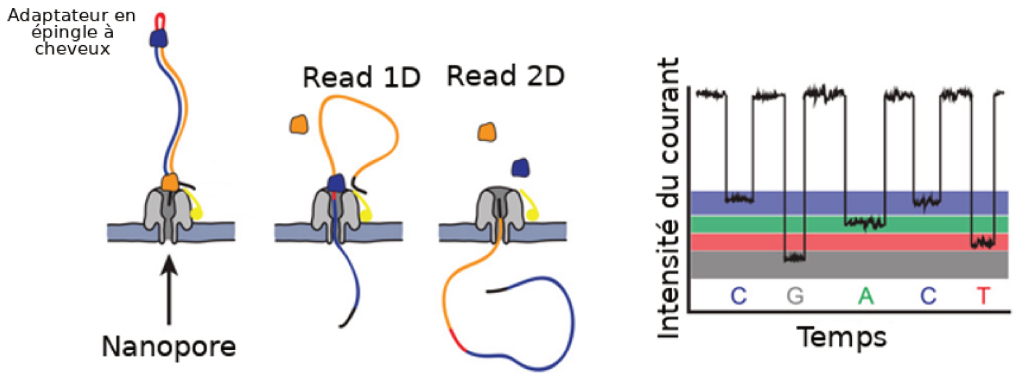

FIGURE 2.9 – **Séquençage ONT.** Figure adaptée de [93].

de couvrir 200 fois le génome. On notera alors qu'une telle expérience représente une couverture de 200x du génome.

Dans la suite du manuscrit, les *reads* de deuxième génération seront qualifiés de *reads courts*, et ceux de troisième génération seront qualifiés de *reads longs*. Le mot-valise *indel* sera utilisé pour faire référence aux erreurs d'insertions et de délétions.

2.3.5 Simulation de *reads*

En addition des technologies de séquençage présentées précédemment, des outils permettant de simuler des *reads*, appelés *simulateurs*, existent également. De tels outils permettent de contrôler précisément la couverture, la longueur des *reads* ainsi que les taux et les types d'erreurs qu'ils contiennent, en se basant sur un génome de référence. En particulier, les régions originales du génome de référence d'où proviennent les *reads*, peuvent ainsi être connues. Ces données simulées sont alors particulièrement intéressantes afin d'évaluer l'impact de ces différentes propriétés sur les résultats d'analyses, et d'identifier les limitations des différents outils. En particulier, elles peuvent ainsi être utilisées afin de guider le développement de nouveaux outils et d'évaluer précisément ces derniers.

Différents simulateurs existent ainsi, afin de simuler le plus précisément possible le comportement des différentes technologies de séquençage. ART [43] permet par exemple de simuler des *reads* Roche / 454, des *reads* ABI / SOLiD et des *reads* Illumina. CuReSim [15] permet de simuler des *reads* des trois technologies précédentes, mais également des *reads* Ion Torrent. MetaSim [94] permet quant à lui de simuler des *reads* Illumina, des *reads* Roche / 454, mais également des *reads* Sanger. Pour les technologies de séquençage de troisième génération, SimLoRD [109] et NPBSS [118] permettent de simuler des *reads* PacBio, tandis que NanoSim [125] et DeepSimulator [73] permettent de simuler des *reads* ONT.

2.3.6 Formats des données

Les séquences d'ADN, qu'elles décrivent un génome complet ou soient issues d'une expérience de séquençage, sont stockées dans des fichiers texte de différents formats. Les plus fréquents d'entre eux sont les formats FASTA et FASTQ [21]. Le format FASTA est le plus simple des deux, et contient deux types d'information pour

```
>M2:A5KR6:1:1101:21832:1000/1
CAGACTTACGGCGGCGCTAATCGCTGCGATATTAACTGTATAACAGCCAA
>M2:A5KR6:1:1101:21832:1000/2
ACGATGATATGGCAGATTTATTAATAAAAAAACCAGTGAGTGAGTTAATC
```

FIGURE 2.10 – **Deux *reads* d'un fichier au format FASTA.** Le «/1» et le «/2» à la fin des *headers* signifie que les deux *reads* sont pairés. Chaque *read* est ici décrit sur deux lignes.

```
@M2:A5KR6:1:1101:21832:1000/1
CAGACTTACGGCGGCGCTAATCGCTGCGATATTAACTGTATAACAGCCAA
+
=CBCCGGGGGG[G]GGGGGGGFGGGGGGGGGGGGFGG0EGGGGGGFGGGGCG
@M2:A5KR6:1:1101:21832:1000/2
ACGATGATATGGCAGATTTATTAATAAAAAAACCAGTGAGTGAGTTAATC
+
CCCCCFG;<GGGGGGGGGGG/GGGGGGGGGGGGGGGGAGGGGGFGGGG>G
```

FIGURE 2.11 – **Deux *reads* d'un fichier au format FASTQ.** Le «/1» et le «/2» à la fin des *headers* signifie que les deux *reads* sont pairés. Chaque *read* est ici décrit sur quatre lignes. La quatrième et la huitième ligne décrivent les scores Phred des *reads*. Par exemple, la lettre G, représentée en gras dans la quatrième ligne de la première séquence, correspond à un score de qualité Phred de 38, et donc à une probabilité d'erreur d'identification de 0,00016%.

chaque séquence. La première information est un en-tête (*header*), commençant par le caractère «>», et contenant des informations relatives à la séquence, typiquement, au moins, un identifiant unique. Ce *header* est généralement décrit sur une seule ligne. À la suite de ce *header*, la séquence en elle-même est décrite, sous forme d'une chaîne de caractères sur l'alphabet ADN, pouvant s'étendre sur une ou plusieurs lignes. Un exemple de fichier au format FASTA est illustré Figure 2.10. Le format FASTQ définit, quant à lui, le premier caractère des *headers* par un «@». De plus, ce format décrit également des informations supplémentaires pour chaque séquence, alors décrite sur un ensemble de quatre lignes. En particulier, ce format décrit un score de qualité, appelé score Phred [26, 27], pour chaque base de la séquence. Plus précisément, ce score Phred, noté Q, permet d'estimer la probabilité P d'erreur d'identification d'une base. En effet, ces deux valeurs sont interdépendantes, et respectent les équations $Q = -10\,log_{10}\,P$ et $P = 10^{\frac{-Q}{10}}$. Un encodage ASCII de ce score de qualité est ainsi stocké au sein des fichiers FASTQ. Un exemple de fichier au format FASTQ est illustré Figure 2.11.

2.3.7 Résumé des technologies disponibles

L'ensemble des technologies présentées précédemment, ainsi que les plateformes de séquençage le plus répandues de ces technologies, sont résumées Table 2.2. Cette table décrit notamment le débit, le temps nécessaire au séquençage, le coût et l'année de sortie de ces différentes plateformes, ainsi que la longueur, le nombre, et le taux d'erreurs des *reads* pouvant être générés.

Technologie	Plateforme	Longueur (bp)	*Reads*	Débit (Gb)	Temps	Erreurs (%)	Prix ($) Séquenceur	Prix ($) Gb	Année
Première génération									
Sanger	-	500 - 600	-	-	-	0,001	-	-	1997
Maxam-Gilbert	-	500 - 600	-	-	-	0,001	-	-	1997
Deuxième génération									
454	GS Junior	400 - 600	0,1 M	0,035	10 h	1 (indel)	108 000	40 000	2010
454	GS Junior+	700 - 1 000	0,1 M	0,07	18 h	1 (indel)	108 000	19 500	2014
454	GS FLX Titanium XLR70	450 - 600	1M	0,45	18 h	1 (indel)	450 000	15 500	2009
454	GS FLX Titanium XL+	700 - 1 000	1M	0,7	23 h	1 (indel)	450 000	9 500	2011
Illumina	Genome Analyzer	100	320 M	600	11 jours	1 (subs.)	250 000	400	2006
Illumina	MiniSeq Débit moyen	150	14 - 16 M	2,1 - 2,4	17 h	1 (subs.)	50 000	200 - 300	2013
Illumina	MiniSeq Haut débit	150	44 - 50 M	6,6 - 7,5	24 h	1 (subs.)	50 000	200 - 300	2013
Illumina	MiSeq v2	250	24 - 30 M	7,5 - 8,5	39 h	0,1 (subs.)	99 000	142	2011
Illumina	MiSeq v3	300	44 - 50 M	13,2 - 15	21 - 56 h	0,1 (subs.)	99 000	110	2011
Illumina	NextSeq 500/550 Débit moyen	150	260 M	32 - 40	26 h	1 (subs.)	250 000	40	2014
Illumina	NextSeq 500/550 Haut débit	150	800 M	100 - 120	29 h	1 (subs.)	250 000	33	2014
Illumina	HiSeq2500 v2	250	600 M	125 - 150	60 h	0,1 (subs.)	690 000	40	2012
Illumina	HiSeq2500 v3	100	3 G	270 - 300	11 jours	0,1 (subs.)	690 000	45	2012
Illumina	HiSeq2500 v4	125	4 G	450 - 500	6 jours	0,1 (subs.)	690 000	30	2012
Illumina	HiSeq4000	150	2,5 G	650 - 750	1 - 3,5 jours	0,1 (subs.)	900 000	22	2015
Illumina	HiSeq X	150	2,6 - 3 G	800 - 900	< 3 jours	0,1 (subs.)	1 000 000	7	2016
SOLiD	5500 Wilfire	50 - 75	700 M	160	6 jours	0,1 (subs.)	349 000	130	2011
SOLiD	5500xl	50 - 75	1,4 G	320	10 jours	0,1 (subs.)	595 000	70	2013
Ion Torrent	PGM 314	200	400 - 550 k	0,06 - 0,1	3,7 h	1 (indel)	49 000	25 - 1 000	2011
Ion Torrent	PGM 316	400	2 - 3 M	0,6 - 1	4,9 h	1 (indel)	49 000	700 - 1 000	2011
Ion Torrent	PGM 318	400	4 - 5,5 M	1 - 2	7,3 h	1 (indel)	49 000	450 - 800	2013
Ion Torrent	Proton	200	60 - 80 M	10	2 - 4 h	1 (indel)	224 000	80	2012
Ion Torrent	S5 520	400	3 - 5 M	1,2 - 2	4 h	1 (indel)	65 000	1 200 - 2 400	2015
Ion Torrent	S5 530	400	15 - 20 M	6 - 8	4 h	1 (indel)	65 000	475 - 950	2015
Ion Torrent	S5 540	200	60 - 80 M	10 - 15	2,5 h	1 (indel)	65 000	300	2015
Troisième génération									
PacBio	RS II	20 k	55 000	0,5 - 1	4 h	10 - 15 (indel)	695 000	1 000	2013
PacBio	Sequel	8 - 12 k	350 000	3,5 - 7	0,5 - 6 h	10 - 15 (indel)	350 000	N/A	2016
ONT	MinION	10 - 200 k jusqu'à 1 M [46]	> 100 000	1,5	< 48h	12 - 30 (indel) biais homopolymers	1 000	750	2014
ONT	PromethION	10 - 200 k jusqu'à 1 M [46]	N/A	4 000	N/A	12 - 30 (indel) biais homopolymers	75 000	N/A	2016

TABLE 2.2 – **Caractéristiques des technologies et plateformes de séquençage.** Tableau adapté de [34].

2.4 Point de vue informatique

Dans cette section, nous abordons les données de séquençage d'un point de vue informatique, en décrivant notamment les principales structures de données utilisées pour leur traitement.

2.4.1 Notions de combinatoire des mots

Les séquenceurs produisent donc des fichiers texte, contenant les *reads* sous forme de chaînes de caractères. Rappelons alors quelques notions de base de combinatoire des mots.

Un *mot* w est une suite ordonnée de caractères (ou lettres, ou symboles, ces trois termes seront indifféremment utilisés) sur un alphabet Σ. Comme mentionné précédemment, nous utiliserons ici l'alphabet ADN $\Sigma = \{A, C, G, T\}$. `CATATAG` est par exemple un mot sur l'alphabet Σ. Le *mot vide*, représentant une suite de zéro lettre, est noté ε.

La *longueur* d'un mot w, notée $|w|$ est définie comme le nombre de caractères du mot w. Classiquement, un mot w de longueur k est appelé un *k-mer*.

La lettre du mot w située à l'indice i est notée $w[i]$, $\forall i$ tel que $0 \leq i < |w|$ (la numérotation des indices d'un mot commence en 0). On appelle chaque indice i une *position* sur w. Ainsi, $w = w[0]w[1]...w[|w| - 1]$.

Un mot x est un *facteur* d'un mot w s'il existe deux mots u et v tels que $w = uxv$. Un mot x est un *préfixe* d'un mot w s'il existe un mot v tel que $w = xv$. Un mot x est un *suffixe* d'un mot w s'il existe un mot u tel que $w = ux$. Par exemple, en posant $w =$ `CATATAG`, alors `TATA` est un facteur de w, `CAT` est un préfixe de w, et `AG` est un suffixe de w. Dans la suite du manuscrit, le facteur de x commençant à la position i et se terminant à la position j sera noté $x[i...j]$.

Un mot x est un *sous-mot* d'un mot w s'il existe $|x| + 1$ mots $y_0, y_1, ..., y_{|x|}$ tels que $w = y_0x[0]y_1x[1]y_2...y_{|x|-1}x[|x| - 1]y_{|x|}$. Par exemple, en posant à nouveau $w =$ `CATATAG`, alors `CAAG` est un sous-mot de w.

Deux mots w_1 et w_2 sont *conjugués* s'il existe deux autres mots u et v tels que $w_1 = uv$ et $w_2 = vu$. Par exemple, les mots $w_1 =$ `CAACT` et $w_2 =$ `CTCAA` sont conjugués, avec $u =$ `CAA` et $v =$ `CT`.

La *distance de Hamming* entre deux mots w_1 et w_2 de même longueur correspond au nombre de positions pour lesquelles les deux mots diffèrent. Plus généralement, la *distance d'édition*, ou *distance de Levenshtein*, correspond au nombre minimal d'opérations d'édition (substitution, insertion ou délétion) nécessaires pour transformer un mot w_1 en un mot w_2. Les opérations d'insertions et de délétions étant ici autorisées, cette distance peut être calculée sur deux mots de longueurs différentes.

2.4.2 Structures de données

Au vu de l'important volume de données généré par les séquenceurs, des structures de données permettant de stocker (on parlera plutôt d'*indexation*, et on utilisera alors le terme *indexer*) et de traiter les *reads* sont nécessaires. Quelques unes des structures de données les plus fréquemment utilisées en bioinformatique, plus particulièrement les structures d'indexation (aussi simplement appelées *index*) et les graphes, sont présentées ici. De telles structures de données sont cependant très nombreuses, et cette section n'a pas pour vocation d'être exhaustive. Ainsi, seules les structures de données dont il sera fait mention au sein de ce manuscrit sont abordées ici.

2.4.2.1 Index

Le but d'un index est de stocker de manière efficace une séquence y de longueur n, afin de répondre à diverses questions concernant son contenu. Ces questions concernent, classiquement, l'appartenance d'un mot x à la séquence. On peut par exemple chercher à obtenir la position de la première occurrence de x dans y, le nombre d'occurrences de x dans y, ou encore la liste des positions des occurrences de x dans y.

Une structure d'indexation classiquement utilisée est l'*arbre des suffixes* [119]. C'est un arbre permettant de reconnaître l'ensemble des suffixes d'un mot, et dans lequel deux chemins distincts ayant la même origine ont toujours des fins distinctes. Ainsi, chaque chemin partant de la racine de l'arbre dénote un suffixe différent du mot d'origine. Une fois l'arbre construit pour un mot y, il est alors facile de le parcourir afin de rechercher l'information concernant la présence d'un autre mot x au sein de y. L'arbre des suffixes peut être représenté de façon non compacte, où une transition entre deux nœuds est toujours étiquetée par un unique caractère, ou de façon compacte, où une transition entre deux nœuds peut être représentée par un mot. Un exemple d'arbre des suffixes, en version non compacte, est illustré Figure 2.12. Le même arbre des suffixes, en version compacte, est illustré Figure 2.13. Cependant, construire un tel arbre pour de larges génomes consomme d'importantes quantités de mémoire, même en utilisant des implémentations optimisées telles que [59]. En particulier, l'indexation d'un génome humain à l'aide d'un arbre des suffixes demande un espace mémoire de 50 Go.

Afin de réduire cette importante consommation de mémoire, d'autres structures d'indexation ont été développées. C'est le cas par exemple de la *table des suffixes* [80]. Cette table permet de stocker l'ensemble des suffixes d'un mot y de longueur n, dans l'ordre lexicographique, en associant à chacun de ces suffixes sa position de début dans le mot. Tout comme pour l'arbre des suffixes, il est alors possible, une fois la table construite, de la parcourir afin de rechercher l'information concernant la présence d'un mot x au sein de y, via une recherche par dichotomie. Un exemple de table des suffixes est illustré Figure 2.14.

La table des suffixes peut également être *augmentée*. Dans ce cas, la table additionnelle des *LCP* (*Longest Common Prefix*) est également calculée. Cette table permet d'obtenir la longueur du plus long préfixe commun entre deux suffixes situés à des positions consécutives de la table des suffixes. À partir de cette table des LCP, les notions additionnelles de *PSV* (*Previous Smaller Value*) et de *NSV* (*Next Smaller Value*) sont définies comme suit, pour toute position $1 \leq i < n$:

- PSV[i] = max{j tel que $0 \leq j < i$ et LCP[j] < LCP[i]}.
- NSV[i] = min{j tel que $i < j \leq n$ et LCP[j] < LCP[i]}.

Pour chacune de ces notions, une table additionnelle est alors calculée. La table des suffixes augmentée est alors définie comme l'ensemble composé de la table des suffixes, de la table des LCP, et de ces deux tables additionnelles. Un exemple de table des suffixes augmentée est illustré Figure 2.15.

Le *FM-index* [28], couplant une *table des suffixes échantillonnée* à une *transformée de Burrows-Wheeler* (*BWT*, pour *Burrows-Wheeler Transform*) [14] est une structure de données également utilisée dans de nombreux outils de bioinformatique. D'une manière similaire à la table des suffixes, la BWT s'obtient en triant non pas les suffixes, mais les conjugués d'un mot y de longueur n. Une chaîne de caractères de longueur n, représentant la BWT, est alors obtenue en concaténant le dernier caractère de chaque conjugué trié. Une propriété remarquable de la BWT est qu'elle tend

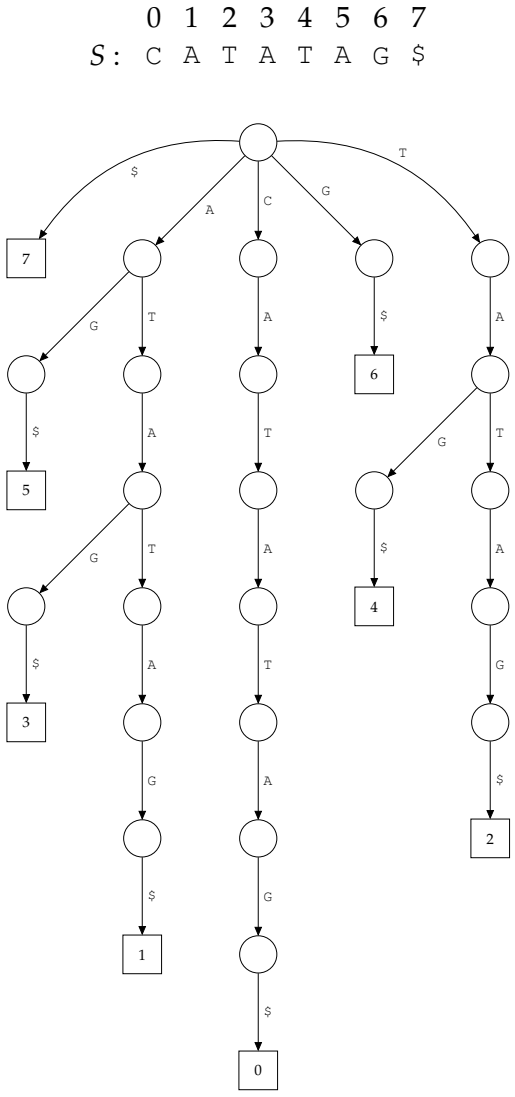

FIGURE 2.12 – **Arbre des suffixes non compact** du mot CATATAG. Un marqueur de fin de mot, $, est ajouté, afin que les suffixes du mots ne puissent pas avoir d'occurrence interne. La construction de l'arbre est ainsi simplifiée. Les feuilles, représentées par des carrés, désignent la fin d'un suffixe, et contiennent la position de début de ce suffixe dans le mot. Les transitions entre les nœuds sont ici étiquetées par un unique caractère.

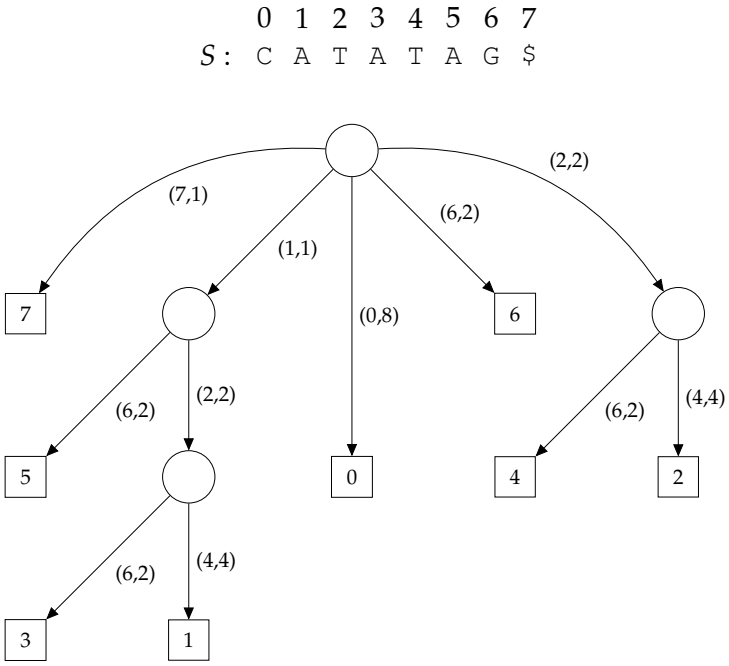

FIGURE 2.13 – **Arbre des suffixes compact** du mot CATATAG. Les feuilles, représentées par des carrés, désignent la fin d'un suffixe, et contiennent la position de début de ce suffixe dans le mot. Les transitions entre les nœuds sont ici étiquetées par un couple (position, longueur) représentant un facteur du mot initial, pouvant être composé de plusieurs lettres.

	0	1	2	3	4	5	6	7
S :	C	A	T	A	T	A	G	$

SA	Suffixes triés
7	$
5	AG$
3	ATAG$
1	ATATAG$
0	CATATAG$
6	G$
4	TAG$
2	TATAG$

FIGURE 2.14 – **Table des suffixes** (notée SA, pour *Sufix Array*) du mot CATATAG. Pour les mêmes raisons que pour l'arbre des suffixes, un marqueur de fin de mot, $, est ajouté. On remarque que la table des suffixes correspond à un parcours en profondeur des arbres des suffixes illustrés Figure 2.12 et Figure 2.13. Cette propriété est liée au fait que, dans ces deux exemples, les transitions entre les nœuds des arbres sont triées par ordre lexicographique.

	0	1	2	3	4	5	6	7
S:	C	A	T	A	T	A	G	$

i	SA	suff[SA[i]]	LCP[i]	PSV	NSV
0	7	$	-1		
1	5	AG$	0	0	8
2	3	ATAG$	1	1	4
3	1	ATATAG$	3	2	4
4	0	CATATAG$	0	0	8
5	6	G$	0	0	8
6	4	TAG$	0	0	8
7	2	TATAG$	2	6	8
8			-1		

FIGURE 2.15 – **Table des suffixes augmentée** du mot CATATAG. Par convention, les valeur de LCP[0] et LCP[$n+1$] sont fixées à -1.

à regrouper les symboles identiques. Cette propriété propriété peut s'avérer particulièrement intéressante pour la compression de données. Un exemple de BWT est donné Figure 2.16. La table des suffixes échantillonné a, quant à elle, la particularité de n'être construite que pour un sous-ensemble des suffixes du mot d'origine. Dans ce cas, lors de l'accès aux parties de la table n'ayant pas été construites, ces dernières sont recalculées au moment de l'exécution, à l'aide de la BWT. Contrairement aux autres structures d'indexation, la recherche d'un mot x au sein du FM-index s'effectue en parcourant x de droite à gauche, via un procédé appelé recherche arrière, ou *backward search*. De plus, cette structure d'indexation est hautement efficace en termes de consommation de mémoire, et permet notamment l'indexation d'un génome humain en utilisant seulement 2 Go de mémoire.

2.4.2.2 Graphes

Les *graphes* sont également largement utilisés en bioinformatique. Rappelons ici quelques notions de base de théorie des graphes.

Un *graphe non orienté* $G = (V, E)$ est un ensemble de sommets $v \in V$ et d'arêtes $e \in E$ telles que $e = \{u, v\}$, $u, v \in V$. Un *graphe orienté* $G = (V, E)$ est un ensemble de sommets $v \in V$ et d'arcs $e \in E$ tels que $e = (u, v)$, $u, v \in V$.

Une arête $\{u, v\}$ d'un graphe non orienté permet donc de se rendre de u vers v, et de v vers u, alors qu'un arc (u, v) d'un graphe orienté ne permet de se rendre que de u vers v. L'arc (u, v) est alors qualifié d'arc *entrant* en v, et d'arc *sortant* de u.

Le nombre de sommets et d'arêtes (resp. d'arcs) sont notés respectivement $|V|$ et $|E|$.

Au sein d'un graphe dirigé, un *embranchement* est défini comme un sommet possédant plus d'un arc sortant.

Un *graphe étiqueté* est un graphe dont les arêtes (resp. les arcs) sont affectées d'étiquettes (mots, lettres, symboles, etc). Un *graphe pondéré* est un graphe étiqueté dont les étiquettes sont des nombres réels, positifs ou nuls. On représentera les arêtes (resp. les arcs) d'un graphe pondéré par $\{u, p, v\}$ (resp. (u, p, v)), où p représente le

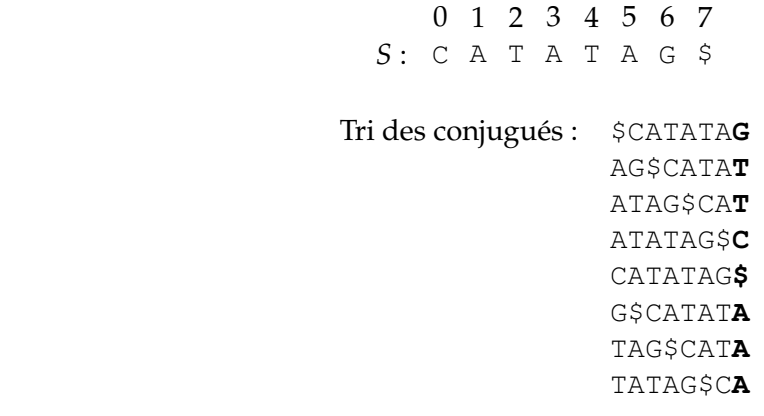

FIGURE 2.16 – **Transformée de Burrows-Wheeler** du mot CATATAG. Comme pour l'arbre et la table des suffixes, un marqueur de fin de mot, \$, est ajouté. Sur cet exemple, on peut remarquer que la BWT a permis de regrouper tous les caractères identiques au sein de différents blocs.

poids de l'arête (resp. de l'arc). Les graphes peuvent également étiquetés ou pondérés en associant des étiquettes ou des poids directement à leurs sommets plutôt qu'à leurs arêtes ou à leurs arcs.

Les graphes peuvent être visités en suivant leurs arêtes (resp. leurs arcs). Un *chemin* c de longueur $n+1$ est un ensemble de sommets $(u_0, \ldots, u_i, \ldots, u_n)$ tels que $\{u_i, u_{i+1}\}$ (resp. (u_i, u_{i+1})) $\in E, \forall i, 0 \leq i < n$.

Un *cycle* (au sein d'un graphe non orienté) ou un *circuit* (au sein d'un graphe orienté) est un chemin tel que $\exists p\,;\, u_p = u_0$.

Au sein d'un graphe pondéré, le *poids d'un chemin* est défini comme la somme des poids des arêtes (resp. des arcs) constituant ce chemin. Le *plus court chemin* entre deux sommets est alors défini comme le chemin de poids minimum parmi l'ensemble des chemins permettant de relier ces deux sommets.

Un *sous-graphe* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ d'un graphe $G = (V, E)$ est un graphe tel que $\mathcal{V} \subseteq V$ et $\mathcal{E} \subseteq E$.

Au sein d'un graphe non orienté G, une *composante connexe* est un ensemble maximal de sommets $\mathcal{V} \subseteq V$ tel que pour toute paire de sommets $\{u, v\} \in \mathcal{V}^2$, il existe un chemin reliant u à v. La maximalité de cet ensemble de sommets signifie qu'aucun autre sommet de V ne peut être ajouté à $\mathcal{V}$ sans invalider la condition.

Au sein d'un graphe orienté, les composantes connexes sont définies sans prendre en compte l'orientation des arcs. On dit alors que $\mathcal{V} \subseteq V$ est une *composante fortement connexe* s'il existe un chemin orienté de tout sommet $u \in \mathcal{V}$ vers tout sommet $v \in \mathcal{V}$.

Un *graphe complet* est un graphe dont toutes les paires de sommets distincts $\{u, v\}$ sont connectées par une arête (resp. par deux arcs, un de u vers v, et un de v vers u).

Au sein d'un graphe non orienté $G = (V, E)$, une *clique* est un ensemble de sommets $\mathcal{V} \subseteq V$ dont le sous-graphe correspondant est complet. Plus précisément, $\mathcal{V}$ est une clique si, pour toute paire de sommets $\{u, v\} \in \mathcal{V}^2$, il existe une arête $e = \{u, v\} \in E$. Une *clique maximale* d'un graphe non orienté est alors une clique possédant le plus grand nombre de sommets, et ne pouvant être étendue en y ajoutant d'autres sommets.

L'un des graphes les plus utilisés en bioinformatique est le *graphe de de Bruijn* (noté *DBG*, pour *De Bruijn Graph*), décrit par Nicolaas Govert de Bruijn dans [13]. De

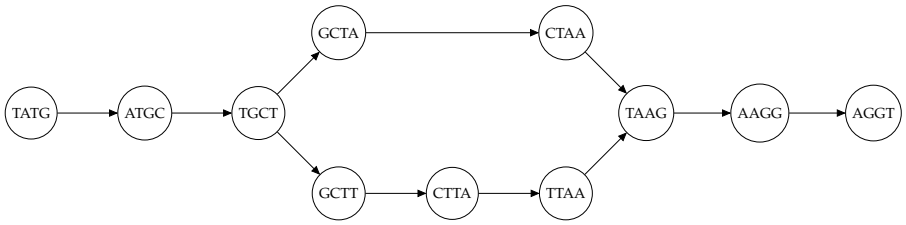

FIGURE 2.17 – **Graphe de de Bruijn d'ordre 4** de l'ensemble de séquences $S = \{$TATGCT, TGCTAA, GCTTAA, TAAGGT$\}$. Afin de simplifier le graphe, les séquences complémentaires inverses des k-mers ne sont pas considérées ici.

tels graphes ont cependant déjà été décrits antérieurement, notamment par Camille Flye Sainte-Marie [96] et Irving J. Good [32]. Le graphe de de Bruijn *d'ordre k* (ou graphe de de Bruijn de *rang k*) d'un ensemble de séquences S est un graphe orienté $G = (V, E)$. L'ensemble des sommets V est défini comme l'ensemble des k-mers de S. Notons $U_k(S)$ cet ensemble. Alors $V = U_k(S)$. L'ensemble des arcs E est défini comme l'ensemble des $e = (u, v)$ tels que $u, v \in V$ et $u[1...k-1] = v[0...k-2]$. En d'autres termes, les arcs représentent des chevauchements préfixe / suffixe de longueur $k-1$ entre les sommets. Un exemple de graphe de de Bruijn est donné Figure 2.17. Un graphe de de Bruijn peut contenir des *bulles* et des *pointes*. Les bulles représentent des régions divergentes entre des séquences similaires, pouvant être causées par des erreurs de séquençage, et se manifestent sous forme de chemins alternatifs. Les pointes, quant à elles, sont également dues à la présence d'erreurs de séquençage, et amènent vers des sommets à partir desquels aucun chemin ne peut être suivi. Un graphe de de Bruijn comportant une bulle et une pointe est illustré Figure 2.18.

D'une manière plus générale, un *graphe de chevauchement* $G = (V, E)$ est un graphe orienté et pondéré, permettant de représenter tous les chevauchements préfixe / suffixe entre les séquences d'un ensemble de séquences S. L'ensemble des sommets V est ici directement défini comme l'ensemble des séquences $s \in S$. L'ensemble des arcs E est défini comme l'ensemble des $e = (u, \ell, v)$ tels que $u, v \in V$ et $u[|u| - \ell ... |u| - 1] = v[0...\ell - 1]$. En d'autres termes, les arcs représentent des chevauchements préfixe / suffixe entre les séquences, et la longueur de ces chevauchements définit le poids des arcs. Un exemple de graphe de chevauchement est donné Figure 2.19.

Un autre type de graphe, appelé *graphe orienté acyclique* (noté *DAG*, pour *Directed Acyclic Graph*) est également fréquemment utilisé, notamment pour représenter des alignements multiples de séquences, comme présenté Section 2.5.1. Comme son nom l'indique, un tel graphe est un graphe orienté qui ne comporte aucun cycle. Par exemple, le graphe de la Figure 2.22 est un DAG. Bien que de tels graphes ne soient, en général, pas des DAG, les graphes de de Bruijn de la Figure 2.17 et de la Figure 2.18, ainsi que le graphe de chevauchement de la Figure 2.19, sont également des DAG.

2.5 Problématiques

La démocratisation du séquençage, et l'important volume de données généré, ont permis l'étude de nombreuses questions dans le domaine de la biologie. Ces questions incluent notamment la recherche de similarités entre séquences, afin de

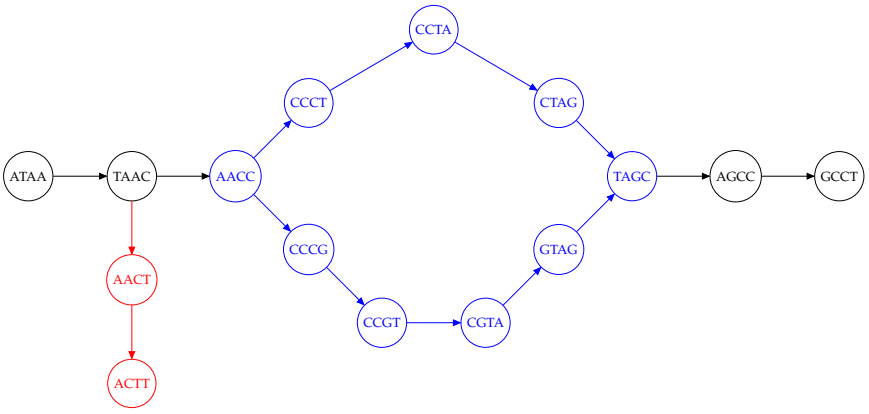

FIGURE 2.18 – **Une bulle et une pointe au sein d'un graphe de de Bruijn.** La bulle est représentée en bleu, et la pointe en rouge.

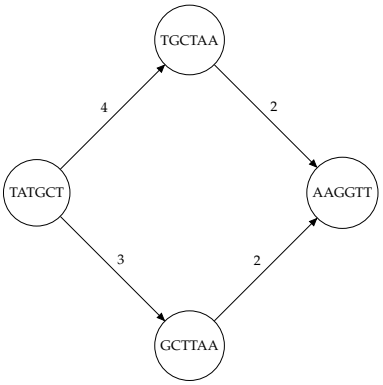

FIGURE 2.19 – **Graphe de chevauchement** de l'ensemble de séquences $S = \{$TATGCT, TGCTAA, GCTTAA, AAGGTT$\}$. Les étiquettes des arcs représentent la longueur des chevauchements entre les séquences associées aux sommets. Pour plus de clarté, les séquences complémentaires inverses et les chevauchements de longueur 1 ne sont pas représentés.

```
    0 1 2 3 4 5
S1: A T T G - A
S2: A T - C C A
```

FIGURE 2.20 – **Alignement de deux séquences** $S_1 = \texttt{ATTGA}$ et $S_2 = \texttt{ATCCA}$. Les positions 0, 1 et 5 représentent des correspondances, ou *matches*, entre les séquences. La position 3 représente une substitution entre les séquences. La position 2 représente une délétion dans S_2, ou l'insertion d'un T dans S_1. La position 4 représente une délétion dans S_1 ou l'insertion d'un C dans S_2.

mettre en évidence des variations structurelles, des mutations ou encore du polymorphisme, parfois limité à un seul nucléotide (on parle alors de *SNV*, pour *Single Nucleotide Variation*). L'assemblage est également une importante problématique liée au séquençage. En effet, des génomes de référence ne sont pas disponibles pour l'ensemble des espèces. De plus, les *reads* produits par les séquenceurs sont bien plus courts que les chromosomes dont ils sont initialement issus, pour une grande majorité d'espèces. L'assemblage vise ainsi à combiner ces *reads*, afin de recréer les chromosomes originaux, et de déterminer les séquences d'ADN de nouveaux génomes. Enfin, les erreurs de séquençage constituent un frein à ces différentes études. Ainsi, la correction de ces erreurs est également une problématique majeure directement liée à l'émergence des technologies de séquençage. Cette section présente plus en détail certaines de ces problématiques, qui seront abordées dans ce manuscrit, et plus particulièrement l'alignement de séquences, l'assemblage, et la correction d'erreurs, constituant le cœur des travaux menés durant cette thèse.

2.5.1 Alignement

L'*alignement* de deux séquences x et y est un procédé consistant à trouver le nombre minimum d'opérations d'édition (substitution, insertion, délétion) permettant de transformer x en y. Ce procédé est au cœur de la bioinformatique, et permet de retrouver des similarités entre les séquences. En fonction de ces similarités, il est alors possible de déterminer des origines communes entre les séquences, et de mettre en avant, par exemple, des gènes identiques, ou des organismes ayant un ancêtre commun. L'alignement peut être réalisé de façon *globale*, afin de déterminer la similarité de deux séquences sur l'intégralité de leur longueur, ou de façon *locale*, afin de mettre en avant des régions similaires au sein de ces séquences. Dans les deux cas, un *score* décrivant la similarité ou le pourcentage d'identité entre les deux séquences, peut être calculé. Dans le cas de l'alignement global, une distance entre les deux séquences peut également être calculée. Un alignement est généralement représenté par un mot sur l'alphabet $(\Sigma \cup \{\text{-}\}) \times (\Sigma \cup \{\text{-}\}) - \{(\text{-},\text{-})\}$. Un exemple d'alignement entre deux séquences est donné Figure 2.20.

Le premier algorithme décrivant une solution permettant de réaliser l'alignement global de deux séquences a été introduit par Needleman et Wunsch [91], et repose sur la programmation dynamique. Concernant l'alignement local, le premier algorithme a été décrit par Smith et Waterman [108], et repose également sur la programmation dynamique. Dans les deux cas, l'alignement est réalisé à l'aide d'une matrice à deux dimensions, chaque dimension représentant une des deux séquences. Le résultat est alors obtenu en temps quadratique $\mathcal{O}(MN)$, avec M et N étant la longueur des séquences.

Ces algorithmes ont bénéficié d'optimisations permettant de réduire le nombre d'opérations, telles que le calcul de l'alignement uniquement à l'intérieur d'une bande diagonale au sein de la matrice. Cependant, ils restent extrêmement coûteux en pratique, notamment quand de nombreux alignements doivent être calculés, ou quand les séquences à aligner sont longues et divergentes, comme c'est le cas des *reads* longs.

Ainsi, en pratique, de nombreuses heuristiques sont utilisées au sein des méthodes d'alignement, afin de réduire les temps de calcul. Ces méthodes d'alignement sont appelées des *aligneurs*, ou des *mappeurs*. Les notions d'*alignement* et de *mapping* peuvent en effet être différenciées. Comme indiqué précédemment, l'alignement de deux séquences x et y permet de produire un score, décrivant la similarité entre ces séquences, mais fournit un ensemble d'opérations d'édition permettant de transformer x en y. Le *mapping*, quant à lui, fournit uniquement l'information d'une correspondance (on appellera cette correspondance un *match*) entre les séquences, ainsi que les positions de ce *match*, sans fournir de détails concernant la correspondance base à base entre les séquences. Nous utiliserons cependant sans distinction le terme d'alignement pour parler de la comparaison de séquences. De plus, dans le contexte de l'alignement de séquences, il peut être intéressant d'appliquer moins de contraintes lors de l'alignement des extrémités des séquences. En effet, certaines technologies, particulièrement Illumina, introduisent des erreurs systématiques aux extrémités des *reads*, diminuant ainsi la qualité dans ces régions. Ce procédé de réduction des contraintes aux extrémités des séquences est appelé le *clipping*.

Ces méthodes permettent ainsi de calculer des alignements entre des *reads* et des génomes de référence, ou entre des *reads* entre eux. Les termes de *target* et de *query* sont généralement employés, pour définir respectivement la cible sur laquelle un *read* s'aligne, et ce *read* lui-même. Pour des raisons d'efficacité, la plupart de ces méthodes utilise une structure d'indexation pour stocker la ou les cibles de l'alignement, et alignent ensuite les *reads* en utilisant cet index. Pour cela, une stratégie appelée *seed and extend* est régulièrement utilisée. Premièrement, un ensemble de régions hautement similaires, partagées entre les séquences à aligner, est identifié. Ces régions sont appelées des graines (*seeds*). Classiquement, ces graines peuvent être des chaînes de k-mers, contigus ou espacés (ces k-mers espacés étant des k-mers contenant des positions jokers, permettant de capturer sans distinction un *match* ou une substitution) partagés entre les séquences. Elles peuvent également être des *maximal exact matches* (*MEM*), représentant des *matches* parfaits entre les séquences, ne pouvant être étendus ni à droite ni à gauche. Une fois calculées, ces graines sont ensuite utilisées pour étendre (*extend*) l'alignement, généralement à l'aide de méthodes reposant sur la programmation dynamique.

Parmi les méthodes d'alignement les plus utilisées peuvent notamment être citées BLAST [2], BWA [71] et ses variantes, BWA-SW [70] et BWA-MEM [67], ou encore Bowtie [62] / Bowtie2 [63]. Cependant, les *reads* de troisième génération, de par leur longueur et leur taux d'erreurs, nécessitent des ajustements algorithmiques spécifiques permettant de prendre en compte ces caractéristiques. De ce fait, des méthodes telles que BLASR [16], Minimap [68] / Minimap2 [69], ou encore DALIGNER [90] sont classiquement utilisées. Ces méthodes diffèrent notamment de par le choix de la structure d'indexation et du type de graines utilisé, et de par certains choix algorithmiques réalisés durant l'alignement.

Les alignements générés par ces outils peuvent être représentés dans différents formats. Les formats les plus utilisés sont les formats SAM (*Sequence Alignment/Map*), représentant les alignements dans un format texte, et sa version équivalente binaire, le format BAM (*Binary Alignment/Map*) [123]. Le format SAM se compose de

onze champs obligatoires, séparés par des tabulations, décrits Table 2.3. Ces champs contiennent notamment les identifiants des deux séquences alignées, les positions de début et de fin de l'alignement sur la séquence *target*, ainsi que les opérations d'édition, dans une chaîne de caractères appelée le code *CIGAR*.

Concernant le *mapping*, différents formats de représentations existent également. Le plus répandu est cependant le format PAF (*Pairwise Alignment Format)*, défini par Minimap. Ce format représente les *matches* dans un format texte, et se compose de douze champs, séparés par des tabulations. Ces champs sont décrits Table 2.4. Ils contiennent notamment les identifiants des deux séquences, ainsi que les positions de début et de fin du *match* sur ces deux séquences.

Le concept d'alignement peut également être généralisé d'une paire à un ensemble de séquences. On parlera dans ce cas d'*alignement multiple*. L'alignement multiple permet ainsi de comparer un ensemble de séquences les unes avec les autres. Cette stratégie peut notamment être utilisée afin d'extraire une séquence consensus à partir d'un ensemble de séquences. Cette séquence consensus est déterminée en choisissant, à chaque position de l'alignement multiple, la base la plus fréquemment représentée au sein de l'ensemble de séquences. Un alignement multiple est généralement représenté par une matrice de taille $M \times N$, avec M la longueur de l'alignement, et N le nombre de séquences. Cependant, il est également possible de représenter un alignement multiple à l'aide d'un DAG. Dans ce cas, les bases de chaque séquence sont représentées par des sommets, et des arcs sont présents entre les sommets représentant des bases consécutives au sein des différentes séquences. Un exemple d'alignement multiple, et de séquence consensus en résultant, est illustré Figure 2.21, pour la représentation par matrice. Le calcul de l'alignement multiple et de la séquence consensus du même ensemble de séquences est illustré Figure 2.22, pour la représentation par DAG. Ce processus d'alignement multiple est notamment largement utilisé par les méthodes de correction d'erreurs de séquençage, lesquelles seront davantage développées dans la Section 2.5.3 et dans les chapitres suivants.

2.5.2 Assemblage

Comme indiqué précédemment, les séquenceurs, même de troisième génération, produisent des *reads* bien plus courts que les chromosomes originaux pour une grande majorité d'espèces. La problématique de l'assemblage consiste donc à reconstruire les chromosomes du génome de l'individu séquencé, à partir des *reads*

Colonne	Description	Type
1	*Header* de la séquence *query*	Chaîne de caractères
2	Drapeau décrivant l'alignement (orientation, *clipping*, etc)	Entier
3	*Header* de la séquence *target*	Chaîne de caractères
4	Position de début de l'alignement sur la séquence *target*	Entier
5	Qualité de l'alignement	Entier
6	Code CIGAR	Chaîne de caractères
7	*Header* du *mate*	Chaîne de caractères
8	Position d'alignement du *mate*	Entier
9	Longueur observée de la séquence *target*	Entier
10	Facteur de la séquence *query* alignée	Chaîne de caractères
11	Score de qualité Phred	Chaîne de caractères

TABLE 2.3 – **Description des champs obligatoires du format SAM.** Tableau adapté de [123].

Colonne	Description	Type
1	*Header* de la séquence *query*	Chaîne de caractères
2	Longueur de la séquence *query*	Entier
3	Position de début du *match* sur la séquence *query*	Entier
4	Position de fin du *match* sur la séquence *query*	Entier
5	Orientation du *match* (+ ou -)	Caractère
6	*Header* de la séquence *target*	Chaîne de caractères
7	Longueur de la séquence *target*	Entier
8	Position de début du *match* sur la séquence *target*	Entier
9	Position de fin du *match* sur la séquence *target*	Entier
10	Nombre de positions *matchant* entre les séquences	Entier
11	Longueur du *match*	Entier
12	Qualité du *match*	Entier

TABLE 2.4 – **Description des champs du format PAF.** Tableau adapté de [68].

	0	1	2	3	4	5
S_1 :	A	T	A	G	–	A
S_2 :	A	C	A	C	C	A
S_3 :	T	C	–	G	C	A
S_4 :		T	A	C	C	A
S_5 :	A	T	–	C	C	T

FIGURE 2.21 – **Représentation par matrice de l'alignement multiple et du consensus de 5 séquences.** La séquence consensus est ici obtenue par un vote majoritaire à chaque position. La base la plus représentée à chaque position est reportée en rouge. La séquence consensus ainsi obtenue est alors ATACCA.

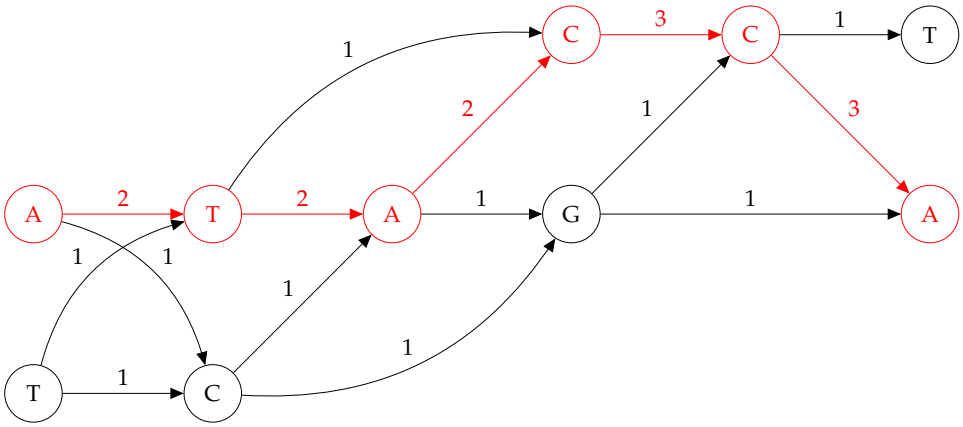

FIGURE 2.22 – **Représentation par DAG de l'alignement multiple et du consensus de 5 séquences.** La séquence consensus est ici obtenue en recherchant le chemin de poids maximal au sein du graphe. Ce chemin est reporté en rouge. La séquence consensus ainsi obtenue est alors ATACCA.

produits par l'expérience. Deux approches d'assemblage peuvent alors être distinguées : *l'assemblage guidé par référence*, et *l'assemblage de novo*. Une méthode permettant de réaliser l'assemblage d'un génome, quelle que soit l'approche employée, est appelée un *assembleur*.

L'assemblage guidé par référence vise à reconstruire un génome, en disposant déjà d'un génome de référence pour l'espèce ou l'individu séquencé. Dans ce cas, les deux génomes sont extrêmement similaires, et le principal intérêt est de mettre en avant des différences entre les deux individus. Ce type d'assemblage est de loin plus aisé que l'assemblage *de novo*, car le génome de référence peut être utilisé afin d'aligner et d'ordonner les *reads* pour produire l'assemblage. Les assembleurs guidés par référence opèrent donc en deux principales étapes. Tout d'abord, les *reads* sont alignés sur le génome de référence. Ensuite, une séquence consensus est calculée entre les *reads* alignés, afin de déterminer la séquence du nouveau génome. Cependant, ce type d'assemblage est entièrement dépendant de la disponibilité d'un génome de référence. Comme mentionné précédemment, des génomes de référence ne sont pas disponibles pour toutes les espèces.

L'assemblage de novo vise ainsi à se détacher de cette contrainte, et à reconstruire un génome à partir des *reads*, sans connaissance *a priori* d'un génome de référence pour l'espèce ou l'individu séquencé. De ce fait, la position originale des *reads* au sein du génome n'est pas connue, et le but est alors de les ordonner les uns par rapport aux autres. Pour cela, les assembleurs *de novo* reposent sur le calcul de chevauchements entre les *reads*. Un *chevauchement* entre deux séquences x et y est défini comme une égalité entre un suffixe de x et un préfixe de y, ou comme une similarité entre ce préfixe et ce suffixe, avec une contrainte sur la distance maximale entre ces deux séquences. Deux séquences partageant un chevauchement significatif (*i.e.*, de longueur suffisante) tendent en effet à indiquer qu'elles sont consécutives dans le génome original. Ainsi, l'assemblage *de novo* consiste en la recherche de chevauchements entre les *reads*, afin de les ordonner et de les combiner, pour créer ainsi de nouvelles séquences, plus longues, et appelées *contigs*. Le principal objectif de l'assemblage *de novo* est alors de produire un contig par chromosome présent dans le génome. Les principaux objets utilisés par les assembleurs *de novo* sont les graphes de chevauchement et les graphes de de Bruijn, introduits dans la Section 2.4.2.2. Ces graphes permettent en effet de représenter de manière élégante les chevauchements entre les *reads*, et ainsi de produire aisément les contigs issus de la combinaison de ces *reads*. Plus précisément, au sein de ces assembleurs, une séquence issue d'un chemin unique du graphe (*i.e.* sans embranchement) est appelée un *unitig*. Un contig est quant à lui obtenu en traversant des embranchements, et donc de multiples chemins uniques, et représente alors une combinaison d'unitigs. Par exemple, le graphe de de Bruijn illustré Figure 2.17 permet d'obtenir l'ensemble d'unitigs $U = \{\texttt{TATGCT}, \texttt{GCTAA}, \texttt{GCTTAA}, \texttt{TAAGGT}\}$, et l'ensemble de contigs $C = \{\texttt{TATGCTAAGGT}, \texttt{TATGCTTAAGGT}\}$.

Tout comme pour les méthodes d'alignement, les propriétés des *reads* courts et des *reads* longs ont amené à des développements algorithmiques différents, prenant en compte les caractéristiques particulières de ces *reads*. Ainsi, parmi les méthodes d'assemblage classiquement utilisées pour les *reads* courts peuvent êtres citées Celera [89], ABySS [47] / ABySS 2.0 [45], SPAdes [5], ou encore SOAPdenovo [72]. Concernant les *reads* longs, Canu [57], Miniasm [68], Falcon [18] et Smartdenovo [1] font partie des assembleurs les plus utilisés à l'heure actuelle. Dans la suite de ce

1. https://github.com/ruanjue/smartdenovo

manuscrit, les simples termes d'assemblage et d'assembleur seront utilisés pour faire référence à l'assemblage *de novo*.

Nous introduisons aussi quelques métriques permettant de décrire la qualité d'un assemblage. La taille *N50* représente la longueur du plus long contig tel que les contigs plus longs que ce dernier couvrent 50% de la longueur totale de l'assemblage. La taille *NG50* représente la longueur du plus long contig tel que les contigs plus longs que ce dernier couvrent 50% de la longueur totale du génome. La taille *NGA50* représente la longueur du plus long contig aligné sur le génome, tel que les contigs alignés plus longs que ce dernier couvrent 50% de la longueur du génome. Ces métriques peuvent aisément être généralisées à d'autres valeurs, classiquement les tailles N75/NG75/NGA75 et N90/NG90/NGA90. Lors de la comparaison d'un assemblage à un génome de référence, un *breakpoint* représente une variation structurelle entre l'assemblage et le génome, comme par exemple une large délétion, une large insertion, ou une large inversion.

2.5.3 Correction

La présence d'erreurs de séquençage dans les *reads* complexifie grandement les analyses sous-jacentes. En effet, ces erreurs peuvent impliquer des temps d'alignement plus importants, voire même fausser les résultats d'un alignement, dû à la dissimilarité qu'elles induisent entre les séquences. Ainsi, des mutations, des variations, ou des SNV peuvent être amenées à être erronément confondues avec des erreurs de séquençage. De plus, ces erreurs induisent également des limitations à l'assemblage, en amenant à la présence de chemins alternatifs au sein des graphes, produisant alors des assemblages plus fragmentés, composés de contigs plus nombreux et plus courts.

De ce fait, détecter et corriger ces erreurs représente la première étape de la plupart des expériences d'analyse des données de séquençage. La correction d'erreurs repose sur différentes méthodologies, dépendant principalement du taux et du profil d'erreurs des *reads*. En effet, ces taux et ces profils varient grandement entre les *reads* courts et les *reads* longs. Tandis que les *reads* courts comportent majoritairement des erreurs de substitutions, et affichent des taux d'erreurs avoisinant 1%, les *reads* longs affichent des taux d'erreurs de plus de 10%, la majorité de ces erreurs étant des insertions et des délétions. De ce fait, des développements algorithmiques différents sont également nécessaires pour les méthodes de correction. Ainsi, de nombreuses méthodes visant à corriger ces erreurs, appelées *correcteurs*, ont été développées.

Concernant les *reads* courts, quatre principales approches sont décrites dans la littérature. La première est l'étude la fréquence des k-mers des *reads* (approche appelée *k-mer spectrum*). Elle se base sur l'observation que les k-mers comportant des erreurs de séquençage sont moins fréquents que les k-mers génomiques au sein des *reads*. Ainsi, un seuil est défini, et les k-mers apparaissant plus souvent que ce seuil sont considérées comme corrects, et ne contenant pas d'erreurs. On parlera alors de k-mers *solides*. Les k-mers apparaissant moins souvent que ce seuil, quant à eux, sont considérés comme incorrects. On parlera alors de k-mers *faibles*). Les k-mers solides sont alors utilisés afin de corriger les k-mers faibles. Parmi les méthodes employant cette approche, peuvent notamment être citées Musket [76], Quake [51], et BLESS [41]. Une seconde approche, reposant sur l'utilisation d'un arbre ou d'une table des suffixes, permet de généraliser l'approche *k-mer spectrum* en rendant possible le traitement de plusieurs valeurs de k à la fois. Des méthodes telles que HiTEC [44], SHREC [103], ou encore HybridSHREC [97] se basent sur cette approche. Une troisième approche vise à corriger les *reads* à l'aide d'alignements multiples. Pour cela,

les alignements entre les *reads* sont tout d'abord calculés. Une séquence consensus est ensuite déterminée pour chaque *read*, par un vote majoritaire à chaque position de l'alignement lui étant associé, tel qu'illustré Figure 2.21. Les correcteurs ECHO [48], et Coral [99] sont basés sur une approche par alignement multiple. Enfin, une dernière approche vise à représenter les *reads* comme des modèles de Markov cachés (*HMM*, pour *Hidden Markov Models*), afin d'extraire, pour chacun de ces modèles, une séquence consensus représentant la correction du *read*. Le correcteur PREMIER [128] se base notamment sur cette approche.

Concernant les *reads* longs, les différentes stratégies et approches de correction sont décrites en détail dans l'état de l'art présenté au Chapitre 3.

2.6 Synthèse

Dans ce chapitre, nous avons introduit les principaux fondements biochimiques de l'ADN, en présentant notamment sa composition, sa structure, ainsi que les mécanismes de transcription et de traduction, permettant, respectivement, le passage de l'ADN à l'ARN et de l'ARN aux protéines.

Les différentes technologies de séquençage, des premières découvertes de Sanger, Maxam et Gilbert en 1977, aux technologies les plus récentes telles que Pacific Biosciences et Oxford Nanopore Technologies, permettant le séquençage de *reads* longs, ont également été présentées. Pour chacune de ces technologies, les principes biochimiques sur lesquels reposent le séquençage, ainsi que les principales propriétés des *reads* pouvant être séquencés, notamment leur longueur et leur taux d'erreurs, ont été décrits.

Les données de séquençage ont ensuite été abordées d'un point de vue informatique, en tant que chaînes de caractères. Les principales structures de données informatique permettant leur traitement, notamment les structures d'indexation et les graphes, ont alors été introduites.

Enfin, les trois principales problématiques liées à l'exploitation de ces données, qui seront abordées au sein de ce manuscrit, ont été présentées. La première de ces problématiques est l'alignement, et vise à mettre en évidence des similarités entre les séquences. La deuxième est l'assemblage, et vise à reconstruire le génome de l'organisme séquencé, à partir de l'ensemble de *reads* produits. Enfin, la troisième de ces problématiques est la correction, et vise à détecter et corriger les erreurs de séquençage présentes au sein des *reads*, afin de réduire leur impact sur les résultats d'analyses.

Chapitre 3

Correction de *reads* longs

Sommaire

3.1 Introduction

Comme décrit dans le Chapitre 2, les technologies de séquençage de troisième génération permettent de produire des *reads* bien plus longs que les technologies de deuxième génération. Cependant, ces *reads* longs sont également bien plus bruités. En effet, tandis que les *reads* courts affichent des taux d'erreurs moyens de 1%, les *reads* longs affichent, quant à eux, des taux d'erreurs variant de 10 à 30%. De plus, les erreurs de substitutions sont présentes en grande majorité dans les *reads* courts, Illumina s'étant imposée comme la technologie de séquençage de deuxième génération la plus utilisée, tandis que les erreurs d'insertions et de délétions sont présentes en plus larges proportions dans les *reads* longs. De ce fait, les méthodes de correction développées pour les *reads* courts, étant ciblées pour la correction de faibles taux d'erreurs de substitutions, ne peuvent pas être appliquées de façon satisfaisante aux *reads* longs.

Ainsi, au vu du potentiel de ces *reads* longs pour la résolution de problèmes d'assemblage de génomes longs et complexes, de nombreuses méthodes de correction, spécifiques aux propriétés des *reads* longs, ont été développées. Ces méthodes peuvent être séparées en deux principales approches. La première de ces approches est la *correction hybride*, et utilise des *reads* courts, de meilleure qualité, en complément des *reads* longs, afin de corriger ces derniers. La seconde est l'*auto-correction*, et vise cette fois à corriger les *reads* longs en se basant uniquement sur les informations contenues dans leurs séquences.

Un des intérêts de la correction hybride est que la correction est alors guidée par les *reads* courts. De ce fait, la profondeur de séquençage des *reads* longs n'affecte pas cette approche. Ainsi, des jeux de données composés d'une très faible couverture de *reads* longs peuvent être efficacement corrigés, tant que la couverture des *reads* courts reste suffisante.

Contrairement à la correction hybride, l'auto-correction, ne se base que sur l'information contenue dans les *reads* longs. Des profondeurs de séquençage plus importantes sont alors nécessaires, et l'auto-correction peut ainsi s'avérer inefficace sur des jeux de données composés de très faibles couvertures de *reads* longs. La couverture nécessaire reste toutefois raisonnable, puisqu'il a été montré qu'à partir de 30x, les méthodes d'auto-correction sont capables de fournir des résultats satisfaisants [105].

Ce chapitre présente donc l'état de l'art des méthodes disponibles pour la correction de *reads* longs, en dressant, pour chacune de ces méthodes, un résumé décrivant les méthodologies utilisées. Les détails concernant les performances, aussi bien en termes de consommation de ressources, qu'en termes de qualité des résultats, ne sont cependant pas abordés ici. Des résultats expérimentaux de l'ensemble de ces méthodes de correction, sur divers jeux de données, de couvertures, de taux d'erreurs, et d'organismes différents, sont présentés dans les Chapitres 4, 5, et 6. Un résume des méthodes de correction hybride disponibles et présenté Table 3.1. Un résumé des méthodes d'auto-correction disponibles est présenté Table 3.2.

3.2 Correction hybride

La correction hybride fut la première approche de correction à être décrite dans la littérature. Cette stratégie se base sur un ensemble de *reads* courts et un ensemble de *reads* longs séquencés pour le même individu. Elle vise à utiliser l'information, plus précise, contenue dans les *reads* courts, afin d'accroître la qualité des *reads* longs.

Quatre approches différentes existent actuellement dans le domaine de la correction hybride.

1. L'alignement des *reads* courts sur les *reads* longs. Une fois les *reads* courts alignés, les *reads* longs peuvent en effet être corrigés, en calculant, pour chacun d'eux, une séquence consensus à partir du sous-ensemble de *reads* courts associé. PBcR / PacBioToCA [56], LSC [4], Proovread [38], Nanocorr [33], NaS [78], LSCplus [42], CoLoRMap [39], et HECIL [20] se basent sur cette approche.
2. L'alignement des *reads* longs sur les contigs obtenus par un assemblage des *reads* courts. De la même façon, les *reads* longs peuvent être corrigés en fonction des contigs avec lesquels ils s'alignent, en calculant des séquences consensus à partir de ces derniers. ECTools [66], HALC [6], et MiRCA [49] adoptent cette méthodologie.
3. L'utilisation d'un graphe de de Bruijn, construit à partir de l'ensemble des *k*-mers des *reads* courts. Une fois construit, les *reads* longs peuvent en effet être y être ancrés. Ce dernier peut alors être traversé, afin de trouver des chemins permettant de relier les régions ancrées des *reads* longs, et ainsi corriger les régions non ancrées. LoRDEC [98], Jabba [86], et FMLRC [115] reposent sur cette stratégie.
4. L'utilisation de modèles de Markov cachés. Ces derniers peuvent en effet être utilisés afin de représenter les *reads* longs. Les modèles peuvent ensuite être entraînés à l'aide des *reads* courts, avant d'en extraire des séquences consensus, représentant les *reads* longs corrigés. Hercules [29] se base sur cette approche.

3.2.1 Alignement de *reads* courts sur les *reads* longs

Cette approche fut la première approche de correction de *reads* longs décrite dans la littérature. Elle consiste en une première étape d'alignement des *reads* courts sur les *reads* longs. Cette étape permet de couvrir chaque *read* long avec un sous-ensemble de *reads* courts. Ce sous-ensemble peut alors être utilisé afin de calculer une séquence consensus de haute qualité, et ainsi corriger le *read* long associé. Les différentes méthodes présentées ici adoptent donc cette approche, mais se différencient notamment par les méthodes d'alignement utilisées, ainsi que par les choix algorithmiques permettant de calculer les séquences consensus.

3.2.1.1 PBcR / PacBioToCA, 2012

PBcR peut calculer les alignements à l'aide de l'aligneur développé et inclus dans l'outil, de BLASR, ou de Bowtie. Afin de réduire les temps de calcul, et d'éviter les alignements partageant trop peu de similarités, ces alignements sont calculés uniquement entre les *reads* courts et les *reads* longs partageant un nombre suffisant de *k*-mers. Seuls les *reads* courts s'alignant sur l'intégralité de leur longueur sont ensuite considérés pour la correction des *reads* longs auxquels ils sont associés. Chaque *read* court peut ainsi s'aligner sur plusieurs *reads* longs. En revanche, sur un même *read* long, un *read* court n'est considéré qu'une seule fois, au niveau de la région sur laquelle il s'aligne avec la plus haute identité.

De plus, pour éviter que les répétitions situées sur différents *reads* longs soient toujours couvertes avec le même sous-ensemble de *reads* courts, ces derniers sont répartis sur les répétitions auxquelles ils sont les plus susceptibles d'appartenir, en fonction de l'identité des alignements. Chaque *read* court est ainsi uniquement autorisé à participer à la correction des N *reads* longs sur lesquels il s'aligne avec la plus

haute identité, *N* représentant approximativement la profondeur de séquençage des *reads* longs. Ainsi, les différentes répétitions ne sont couvertes que par les *reads* courts les représentant le mieux.

Enfin, à partir du sous-ensemble de *reads* courts couvrant chaque *read* long, une séquence consensus est générée, à l'aide du module de consensus inclus dans l'assembleur AMOS [92]. Ce module calcule la séquence consensus en plusieurs étapes, de manière itérative. À chacune de ces étapes, tous les *reads* courts sont alignés sur le consensus actuel (celui-ci étant le *read* long lui-même lors de la première étape), afin de générer un alignement multiple. Cet alignement multiple est alors utilisé pour générer une nouvelle séquence consensus, via un vote majoritaire à chaque position. Les itérations s'arrêtent alors quand la nouvelle séquence consensus est identique à celle de l'étape précédente.

PBcR produit ainsi des *reads* longs corrigés *splittés*. En effet, les extrémités des *reads* longs non couvertes par des alignements ne sont pas incluses dans le calcul de la séquence consensus, et ne sont donc pas reportées dans les versions corrigées des *reads* longs. De plus, si des trous de couverture sont repérés le long de la séquence d'un *read* long, la version corrigée de ce dernier est *splittée*, et chaque région couverte est alors reportée indépendamment, afin d'éviter l'introduction de bases incorrectes dans les *reads* longs corrigés.

3.2.1.2 LSC, 2012

LSC commence par compresser les homopolymers présents dans les *reads* courts et dans les *reads* longs. Pour cela, ces homopolymers sont remplacés par une seule occurrence du nucléotide répété. Ainsi, par exemple, la séquence CTTAGGA est compressée en CTAGA. Cette étape de compression est notamment utilisée afin de faciliter l'étape d'alignement. De plus, afin de pouvoir réaliser l'opération inverse, et décompresser les *reads*, un index est également calculé. Cet index est composé de deux tableaux pour chaque *read* compressé, associant la position des homopolymers compressés) la longueur de ces homopolymers au sein du *read* initial. Une fois les *reads* compressés, les *reads* longs sont concaténés, afin de construire des séquences de longueur plus importante, en ajoutant, entre deux *reads* longs consécutifs, une suite de *n* nucléotides N, *n* représentant la longueur moyenne d'un *read* court.

Les *reads* courts sont alors alignés sur l'ensemble de ces nouvelles séquences, obtenues après la concaténation des *reads* longs. Les alignements sont, par défaut, calculés avec Novoalign [1], bien que LSC soit également compatible avec d'autres aligneurs, plus efficaces en termes de consommation de ressources, tels que BWA ou SeqAlto [88]. À partir des informations issues de l'alignement, un sous-ensemble de *reads* courts est alors assigné à chaque *read* long.

Chaque *read* long est alors corrigé, en calculant, à partir des *reads* courts, des consensus locaux sur les régions des alignements définissant des substitutions, des délétions, ou des insertions entre le *read* long et les *reads* courts. De plus, les régions d'homopolymers compressés, qu'elles soient situées sur le *read* long ou sur les *reads* courts, sont également localement corrigées de la même façon. Pour chacune des régions mentionnées ci-dessus, les séquences de tous les *reads* courts couvrant la région sont décompressées temporairement. Un consensus local est alors calculé par vote majoritaire, afin de corriger cette région dans la séquence du *read* long. Les bases du *read* long ne correspondant à aucune de ces régions ne sont ainsi pas modifiées par l'étape de correction. Une fois toutes les régions nécessitant une correction traitées, les régions d'homopolymers compressés du *read* long, non modifiées lors de l'étape

1. http://www.novocraft.com/products/novoalign

de correction, sont décompressées et laissées non modifiées dans les séquences des *reads* longs corrigés.

LSC produit ainsi des *reads* longs corrigés *trimmés*, définis, pour chaque *read* long, comme la séquence du *read* long décompressé, du point le plus à gauche au point le plus à droite, couverts par au moins un *read* court. Cependant, les bases des *reads* longs n'ayant pas pu être corrigées ne sont pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* longs corrigés ne peuvent pas être *splittés* après la correction.

3.2.1.3 Proovread, 2014

Proovread calcule les alignements à l'aide de SHRiMP2 [22]. Bien que cet aligneur soit utilisé par défaut, tous les aligneurs proposant une sortie au format SAM, tels que BWA ou Bowtie / Bowtie2 sont compatibles avec Proovread, et peuvent ainsi être utilisés. Une étape de validation est ensuite appliquée, afin de filtrer les alignements de faible qualité. Pour cela, des scores normalisés selon la longueur des alignements sont utilisés, dans un contexte local, en représentant les *reads* longs comme une série de petites fenêtres (de longueur 20 par défaut) consécutives. Chaque alignement est alors assigné à une de ces fenêtres, en fonction de la position de son centre.

Une fois les alignements répartis sur les différentes fenêtres, seuls les alignements possédant les plus hauts scores de chaque fenêtre sont considérés pour le calcul de la séquence consensus, lors de l'étape suivante. Pour calculer ce consensus, une matrice composée d'une colonne par nucléotide est utilisée pour chaque *read* long. Cette matrice est alors remplie avec les bases des *reads* courts, en fonction des informations contenues dans les alignements sélectionnés. La séquence consensus est alors obtenue par un vote majoritaire sur chaque colonne de la matrice. Si aucune base provenant de *reads* courts n'est disponible pour une colonne donnée, la base présente dans le *read* long est alors conservée. De plus, un score, imitant le score de qualité Phred, est assigné à chaque base du *read* long corrigé. Ce score est défini à partir du support de la base choisie comme consensus, dans la colonne correspondante de la matrice.

Afin de réduire la consommation de ressources et le temps nécessaire à l'étape d'alignement, particulièrement coûteuse lorsqu'elle est réalisée avec une haute sensibilité, Proovread propose une procédure d'alignement et de correction itérative, où la sensibilité des alignements est augmentée à chaque itération. Cette stratégie est composée de trois cycles de pré-correction, et d'une étape de finition. Ainsi, durant les cycles de pré-correction, seul un sous-ensemble de 20, 30, et 50% de l'ensemble de *reads* courts originaux est utilisé, respectivement pour le premier, le deuxième, et le troisième cycle. Lors du premier cycle, les *reads* courts sont alignés rapidement, avec une sensibilité moyenne. Les *reads* longs sont alors pré-corrigés à l'aide des *reads* courts alignés, comme mentionné dans le paragraphe précédent. De plus, les régions des *reads* longs suffisamment couvertes par les *reads* courts sont masquées, et ne sont plus considérées dans les cycles d'alignement et de correction suivants. Deux nouveaux cycles de pré-correction sont ainsi appliqués, en utilisant de plus grands sous-ensembles de *reads* courts, et des sensibilités croissantes. En moyenne, les deux premiers cycles suffisent à masquer plus de 80% des séquences des *reads* longs. Enfin, pour l'étape de finition, les masques sont retirés des *reads* longs pré-corrigés, et l'ensemble complet de *reads* courts est aligné avec une haute sensibilité, afin de réaliser une dernière étape de correction, et ainsi raffiner les pré-corrections précédentes.

Proovread produit ainsi des *reads* longs corrigés, en version *splittée* et en version non *splittée*. L'étape de *split* est en effet indépendante de l'étape de correction, et repose sur les scores de qualité pseudo-Phred, assignés par la correction à chacune des bases des *reads* longs corrigés. Les régions de faible qualité des *reads* longs corrigés peuvent ainsi être identifiées et éliminées, afin de ne conserver que les régions de haute qualité.

3.2.1.4 Nanocorr, 2015

Nanocorr calcule les alignements à l'aide de BLAST. Un ensemble d'alignements, comprenant des alignements corrects, et s'étendant sur presque toute la longueur des *reads* courts, ainsi que des alignements incorrects ou partiels des *reads* courts, est ainsi obtenu. Une première étape de filtrage est alors appliquée à ces alignements, afin de retirer les alignements courts, et entièrement contenus dans un alignement de taille plus importante. Un algorithme de programmation dynamique, basé sur le problème de la plus longue sous-séquence strictement croissante [102], est ensuite utilisé, afin de sélectionner le sous-ensemble optimal d'alignements de *reads* courts couvrant chaque *read* long.

À partir de ce sous-ensemble, une séquence consensus est alors calculée, à l'aide d'un DAG, en utilisant le module de consensus PBDAGCon, décrit Section 3.3.1.1.

Nanocorr produit ainsi des *reads* longs corrigés et non *trimmés*. De plus, les bases des *reads* longs n'ayant pas pu être corrigées ne sont pas reportées dans une casse différente des bases corrigées, pour les raison mentionnées dans la description de PBDAGCon, Section 3.3.1.1. Ainsi, les *reads* longs corrigés ne peuvent pas être *trimmés* ou *splittés* après la correction.

3.2.1.5 NaS, 2015

Contrairement aux méthodes décrites précédemment, réalisant des corrections locales sur les *reads* longs, ou calculant des séquences consensus à partir des alignements des *reads* courts, NaS génère des séquences corrigées pour les *reads* longs à partir d'assemblages de *reads* courts. Ces séquences, qualifiées de *reads longs synthétiques*, sont générées comme suit.

Tout d'abord, les alignements sont calculés à l'aide de BLAT [52] ou de LAST [53], en fonction du mode de correction souhaité (rapide ou sensible, respectivement). Les *reads* courts ainsi alignés sur les *reads* longs définissent alors des graines. Pour chacun des *reads* longs, une stratégie de type *seed and extend* est ensuite employée. Les graines sont alors comparées à l'ensemble des *reads* courts, afin de recruter de nouveaux *reads* courts similaires, permettant de couvrir les régions du *read* long non couvertes par les alignements initiaux. Ce recrutement est réalisé à l'aide de Commet [79], en considérant que deux *reads* courts sont similaires s'ils partagent un nombre suffisant de k-mers non chevauchants.

Une fois les nouveaux *reads* courts recrutés, le sous-ensemble de *reads* courts formé par ces nouveaux *reads* et par les graines est alors assemblé à l'aide de Newbler [113]. Un contig unique est généralement produit, mais dans les régions répétées, quelques *reads* courts provenant de la mauvaise copie de la répétition peuvent être recrutés, et ainsi produire des contigs additionnels ne devant pas être associés au *read* long original. Pour résoudre ce problème, et ne produire qu'un seul contig par *read* long, le graphe des contigs est alors explicitement construit. Ce graphe est un graphe orienté et pondéré, dont les sommets sont les contigs. Un arc est alors présent entre deux sommets si les contigs associés se chevauchent. Les poids sont

cependant associés aux sommets de ce graphe, et sont définis, pour chacun de ces sommets, comme le taux de couverture du contig associé par les *reads* courts ayant servi de graines. Une fois le graphe construit, l'algorithme de Floyd-Warshall [30, 116] est utilisé afin d'en extraire le plus court chemin. Le contig final est alors obtenu en combinant les contigs associés aux sommets traversés par ce chemin.

Enfin, les *reads* courts sont réalignés sur le contig obtenu, afin de vérifier sa consistance. Le contig est alors défini comme la correction du *read* long original, s'il est suffisamment couvert par les *reads* courts.

Contrairement aux méthodes précédentes, ayant tendance à *trimmer* ou à *splitter* les *reads* longs corrigés, NaS tend, à l'inverse, à produire des *reads* longs corrigés plus longs que les *reads* longs originaux. En effet, dû au processus de recrutement des *reads* courts, NaS inclus souvent des *reads* courts situés en dehors des extrémités de la séquence du *read* long original dans le sous-ensemble de *reads* courts à assembler.

3.2.1.6 LSCplus, 2016

LSCplus se base sur le principe de LSC, présenté Section 3.2.1.2, en y apportant quelques modifications d'implémentation et d'optimisation au niveau de l'alignement et de la correction. Ainsi, comme dans LSC, les homopolymers contenus dans les *reads* longs et dans les *reads* courts sont tout d'abord compressés, et remplacés par un unique nucléotide. L'index enregistrant la taille originale des homopolymers est cependant modifié par rapport à LSC. En effet, il n'est ici composé que d'un unique tableau par *read*, enregistrant pour chaque position, compressée ou non, le nombre original de bases dans le *read* initial. De plus, contrairement à LSC, les *reads* longs ne sont pas concaténés en des séquences de longueur plus importante, séparées par des suites de nucléotides N. Les *reads* courts sont donc directement alignées sur les *reads* longs, à l'aide de Bowtie2.

Chaque *read* long peut alors être corrigé, à l'aide du sous-ensemble de *reads* courts y étant associé. Une séquence consensus est ainsi calculée, par vote majoritaire, à chacune des positions du *read* long compressé. Contrairement à LSC, décompressant les régions nécessitant une correction avant de calculer la séquence consensus, la base la plus fréquente à chaque position est ici déterminée avant la décompression. Une fois la base la plus fréquente sélectionnée, la décompression est alors réalisée, en déterminant la taille de l'homopolymer à partir des informations contenues dans les index des *reads* courts s'alignant à cette position. La séquence ainsi obtenue est alors concaténée à la fin de la séquence consensus, construite itérativement à mesure que les bases du *read* long compressé sont traitées. Les bases correspondant à des positions non couvertes du *read* long compressé sont, quant à elles, décompressées sans modification, et concaténées à la séquence consensus.

Contrairement à LSC, LSCplus produit ainsi des *reads* longs corrigés en version *trimmée* mais également en version non *trimmée*. Comme pour LSC, la séquence d'un *read* long *trimmé* est définie du point le plus à gauche au point le plus à droite de ce *read* long, couverts par au moins un *read* court. Tout comme LSC, les bases des *reads* longs n'ayant pas pu être corrigées ne sont cependant pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* longs corrigés ne peuvent pas être *splittés* après la correction.

3.2.1.7 CoLoRMap, 2016

CoLoRMap calcule les alignements à l'aide de BWA-MEM. Un graphe d'alignement est ensuite construit pour chaque *read* long, à partir du sous-ensemble de *reads*

courts y étant aligné. Ce graphe est un graphe orienté et pondéré, dont chaque sommet est défini par l'alignement d'un *read* court sur le *read* long. Un arc est défini entre deux sommets u et v si les *reads* courts associés à ces deux sommets partagent un chevauchement préfixe / suffixe, sur une longueur suffisante, et avec au plus une erreur. Enfin, le poids d'un arc entre un sommet u et un sommet v est défini comme la distance d'édition entre le suffixe du *read* associé v ne faisant pas partie du chevauchement, et le *read* long original.

Chaque composante connexe de ce graphe est alors traitée indépendamment. Pour chacune de ces composantes connexes, un sommet *source*, correspondant au *read* court aligné le plus à gauche de la composante, et un sommet *cible*, correspondant au *read* court aligné le plus à droite, sont définis. L'algorithme de Dijkstra [23] est alors utilisé, afin de trouver le plus court chemin du graphe permettant de relier la source à la cible. Ce chemin, via les *reads* courts associés aux sommets traversés, dicte alors une correction pour la région du *read* long couverte par les alignements contenus dans la composante connexe.

Pour corriger les régions des *reads* longs n'ayant pas été couvertes par des *reads* courts, CoLoRMap utilise l'information portée par les *reads* courts pairés pouvant être produits par les technologies de seconde génération. Des *One-End Anchors* (*OEA*) sont ainsi définis, afin de corriger ces régions. Ces OEA sont des *reads* courts n'ayant pas pu être alignés, mais dont le *mate*, lui, est aligné sur une région bordant la région non couverte. Pour chacune des régions non couvertes d'un *read* long, l'ensemble d'OEA associé peut alors être défini en observant les alignements situés dans les régions bordantes corrigées.

Pour cette seconde étape de correction, les *reads* courts sont de nouveaux alignés sur les *reads* longs corrigés, obtenus à l'étape précédente, à l'aide de BWA. Ces *reads* courts peuvent alors facilement être alignés sur les régions corrigées des *reads* longs, au vu de la faible divergence entre les *reads* courts et ces régions corrigées. Pour chaque région d'un *read* long n'ayant pas été corrigée lors de l'étape précédente, CoLoRMap définit alors un ensemble d'OEA, en observant les alignements situés dans les régions bordantes. Un assemblage de ces OEA est alors réalisé, à l'aide de Minia [17], afin d'obtenir des contigs, utilisés afin de corriger cette région non couverte.

CoLoRMap produit ainsi des *reads* longs corrigés et non *trimmés*. Cependant, les bases ayant été corrigées sont reportées en majuscule, et les bases n'ayant pas pu être corrigées sont reportées en minuscule. Ainsi, les *reads* longs corrigés peuvent facilement être *trimmés* ou *splittés* après la correction, afin d'en éliminer les régions non corrigées.

3.2.1.8 HECIL, 2018

HECIL calcule les alignements à l'aide de BWA-MEM. À partir des alignements obtenus, les positions erronées, dénotant des désaccords (indel ou substitution) entre un *read* long et les *reads* courts y étant alignés sont marquées. Pour chacune de ces positions, l'ensemble des *reads* courts alignés est alors étudié, et une correction en deux étapes est appliquée.

Tout d'abord, en cas de fort consensus (*i.e.* au moins 90%) des *reads* courts à une position, la base du *read* long est remplacée par la base consensus des *reads* courts. Cette première étape est inspirée des méthodes reposant sur une approche par vote majoritaire, et permet de réaliser une première phase de correction rapide. De plus, elle permet d'éviter de réaliser des corrections erronées à certaines positions, faites

en fonction de *reads* courts fréquents, mais de mauvaise qualité. Les corrections réalisées lors de cette étape permettent également de réduire le nombre de positions à traiter lors l'étape suivante.

Pour les positions erronées restantes, un score est associé à chaque *read* court aligné, en combinant la qualité du *read* court, obtenu via son score Phred, et le score de similarité de l'alignement entre le *read* court et le *read* long. La motivation primaire de ce score combiné est que l'incorporation de la qualité des *reads* courts dans la correction de *reads* longs n'avait été abordée par aucune méthode de l'état de l'art jusqu'alors. Pour chaque position erronée, la base du *read* court maximisant le score combiné est alors choisie comme correction de la base du *read* long. En cas de conflit entre plusieurs *reads* courts, la base du *read* court ayant la plus haute qualité est alors choisie.

De plus, HECIL définit également une méthodologie d'apprentissage itératif. Ainsi, les deux étapes de correction décrites précédemment sont appliquées itérativement. À chaque itération, les scores combinés définis précédemment sont utilisés afin de définir un sous-ensemble de positions à corriger de haute confiance. Seules les positions contenues dans ce sous-ensemble sont donc corrigées lors de cette itération. Ces positions sont alors marquées et fixées, afin de ne plus être modifiées lors des itérations suivantes. L'objectif recherché en ne corrigeant que le meilleur sous-ensemble de positions à chaque itération est d'améliorer les corrections apportées par les itérations suivantes, pouvant bénéficier des *reads* longs mis à jour par les itérations précédentes. Ainsi, lors de l'itération suivante, les *reads* courts sont réalignés sur les *reads* longs mis à jour, et les deux phases de correction sont de nouveau appliquées sur un nouveau sous-ensemble de positions de haute confiance. Les itérations sont stoppées lorsque la variation du nombre de k-mers uniques des *reads* longs, entre deux itérations, est plus faible qu'un seuil donné, ou qu'un nombre maximal d'itérations est atteint.

HECIL produit ainsi des *reads* longs corrigés et non *trimmés*. En effet, des corrections locales sont réalisées sur les *reads* longs, aux positions dénotant des désaccords avec les *reads* courts alignés. Ainsi, les extrémités des *reads* longs, non couvertes par des alignements, ne sont simplement pas modifiées. Cependant, les bases des *reads* longs non couvertes, et n'ayant pas pu être corrigées, ne sont pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* longs corrigés ne peuvent pas être *trimmés* ou *splittés* après la correction.

3.2.2 Alignement de *reads* longs et d'assemblages de *reads* courts

Au vu de la longueur des *reads* courts, ces derniers peuvent être difficiles à aligner sur les régions répétées ou sur les régions fortement bruitées des *reads* longs. Cette approche vise à remédier à ce problème, en commençant par assembler les *reads* courts. En effet, les contigs obtenus par assemblage sont bien plus longs que les *reads* courts. De ce fait, ils peuvent alors plus aisément couvrir les régions répétées ou hautement erronées des *reads* longs, en utilisant le contexte de leurs régions voisines lors de l'alignement. Comme précédemment, les contigs alignés avec les *reads* longs peuvent alors être utilisés afin de calculer des séquences consensus de haute qualité, et de corriger les *reads* longs associés. Les méthodes présentées ici adoptent donc cette approche, et se différencient par les méthodes d'alignement utilisées, et par leurs choix algorithmiques lors du calcul des séquences consensus.

3.2.2.1 ECTools, 2014

ECTools nécessite un ensemble de *reads* courts assemblés en unitigs comme base pour la correction des *reads* longs. Cela signifie que l'assemblage doit être réalisé en amont du lancement d'ECTools, et qu'il n'est donc pas automatiquement généré durant le processus de correction. L'assemblage peut ainsi être produit par n'importe quel assembleur, bien que Celera soit recommandé, de par le fait qu'il garantisse que chaque *read* court soit inclus dans un unitig, en créant, au besoin, des unitigs composés d'un unique *read* court. Cela permet à ECTools de s'assurer que toute l'information portée par les *reads* courts sera transmise à l'ensemble des unitigs, et qu'aucune information ne sera perdue durant le processus d'assemblage.

Les unitigs sont alors alignés sur les *reads* longs à l'aide de l'aligneur Nucmer, disponible dans la suite d'outils MUMmer [60]. Un sous-ensemble d'unitigs couvrant le mieux chaque *read* long est ensuite choisi et utilisé pour la correction. Cet ensemble optimal d'unitigs est déterminé à l'aide d'un algorithme reposant sur le problème de la plus longue sous-séquence strictement croissante, lui aussi disponible dans la suite d'outils MUMmer. Une fois calculé, ce sous-ensemble d'unitigs est alors utilisé pour corriger le *read* long. Les bases du *read* long en désaccord avec les unitigs sélectionnés sont identifiées, et le *read* long est ainsi corrigé, en fonction des bases contenues dans ces unitigs.

ECTools produit ainsi des *reads* longs corrigés *splittés*. Les extrémités des *reads* longs non couvertes par des unitigs sont d'abord éliminées des séquences des *reads* longs corrigés. Ensuite, les régions internes, non couvertes par des unitigs, sont considérées comme non corrigées, et étiquetées avec un taux d'erreurs de 15%. Les régions couvertes sont, quant à elles, considérées comme corrigées, et étiquetées avec un taux d'erreurs de 1%. Le taux d'erreurs moyen de chaque *read* long corrigé est alors calculé sur toute la longueur du *read*, en fonction de ces étiquettes, puis comparé à un seuil de qualité minimale. Si la qualité d'un *read* long corrigé est en dessous ce seuil, le *read* est alors *splitté*, en retirant la plus longue région non corrigée qu'il contient. Ce processus est alors appliqué récursivement à chaque fragment ainsi obtenu, jusqu'à ce que ces derniers soient tous au dessus du seuil minimal de qualité.

3.2.2.2 HALC, 2017

Tout comme ECTools, HALC nécessite également un ensemble de *reads* courts assemblés comme base pour la correction des *reads* longs. L'assemblage doit donc, une fois encore, être réalisé en amont du lancement de HALC, et n'est pas généré automatiquement durant le processus de correction. Contrairement à ECTools, HALC utilise cependant les contigs générés par l'assemblage, plutôt que les unitigs. Les *reads* longs sont ainsi alignés sur les contigs, à l'aide de BLASR, afin d'amorcer le processus de correction.

Un graphe non orienté est ensuite construit à partir des alignements entre les *reads* longs et les contigs, afin que les alignements alternatifs, en particulier au sein des régions répétées, puissent être représentés par différents chemins. Les sommets de ce graphe sont ainsi définis à partir des régions des contigs sur lesquelles un *read* long a été aligné. Une arête est alors présente entre deux sommets si au moins une paire de régions adjacentes, au sein d'un *read* long, est alignée sur les deux régions des contigs correspondant à ces sommets.

Ce graphe est également pondéré, le poids des arêtes étant fixé en fonction de deux règles. Tout d'abord, le poids d'une arête (u, v) est fixé à 0 si les régions correspondant aux sommets u et v sont adjacentes au sein d'un des contigs. Au contraire, pour une arête (u, v) dont les régions correspondant aux sommets u et v ne sont pas adjacentes au sein d'un contig, le poids est fixé à $max\{C_0 - C, 0\}$. C_0 représente la couverture des contigs par les *reads* longs, tandis que C représente le nombre de régions adjacentes des *reads* longs alignées sur les régions associées à u et à v.

Une fois le graphe construit, chaque *read* long est corrigé indépendamment. Pour cela, les chemins représentant ses alignements alternatifs sont recherchés au sein du graphe. Le chemin de poids minimal est alors calculé par programmation dynamique, et les régions des contigs correspondant aux sommets traversés sont utilisées pour corriger le *read* long.

Cependant, en utilisant cette approche, les régions adjacentes des *reads* longs, corrigées avec des régions de contigs non adjacentes, sont très susceptibles d'être des répétitions, et d'avoir été corrigées avec des répétitions similaires plutôt qu'avec les réelles régions du génome auxquelles elles appartiennent. Ces régions sont alors enregistrées, et leur correction est raffinée à l'aide de LoRDEC, un autre outil de correction hybride, présenté Section 3.2.3.1.

HALC produit ainsi des *reads* longs corrigés et non *trimmés*. Cependant, la dernière étape de correction étant basée sur LoRDEC, les bases corrigées sont reportées en majuscules, et les bases non corrigées sont reportés en minuscules, comme indiqué Section 3.2.3.1. Ainsi, les *reads* longs corrigés peuvent facilement être *trimmés* ou *splittés* après la correction. Les *reads* non *trimmés*, les *reads trimmés*, et les *reads splittés* sont donc tous reportés par HALC.

3.2.2.3 MiRCA, 2018

Contrairement à toutes les méthodes présentées jusqu'alors, MiRCA commence par appliquer une étape de filtration sur les *reads* courts. En effet, bien qu'étant de meilleure qualité que les *reads* longs, les *reads* courts contiennent également des erreurs de séquençage, pouvant amener à des imprécisions lors de l'assemblage et de l'alignement des *reads* longs sur les contigs ainsi obtenus. Ainsi, les *reads* courts contenant de trop nombreuses erreurs sont retirés de l'ensemble de *reads* courts, et ne sont pas considérés lors des étapes suivantes. Cette étape de filtration repose sur la fréquence des *k*-mers apparaissant dans les *reads* courts. Les *reads* courts contenant de trop nombreux *k*-mers faibles sont ainsi filtrés.

Les *reads* courts de haute qualité sont alors assemblés à l'aide de SPAdes, un assembleur basé sur un graphe de de Bruijn, permettant de s'assurer que toute l'information contenue dans les *reads* courts sera préservée dans les contigs produits. Ces contigs sont ensuite alignés sur les *reads* longs, à l'aide de BLAST+ [77]. Les *reads* longs peuvent alors être corrigés à partir des alignements obtenus. Dans le cas où un unique contig est aligné sur une région d'un *read* long, la région de ce *read* long est alors simplement remplacée par la séquence du contig aligné. Dans le cas où plusieurs contigs sont alignés sur une même région d'un *read* long, ces alignements définissent alors un alignement multiple, et une approche par vote majoritaire est adoptée pour corriger la région du *read* long. Si le vote majoritaire ne permet pas de mettre en avant une base consensus au sein des contigs, la base présente dans le *read* long à cette position est alors conservée.

MiRCA produit ainsi des *reads* longs corrigés *trimmés*. En effet, les extrémités des *reads* longs, sur lesquelles aucun contig n'est aligné, ne peuvent être traitées et sont donc éliminées des séquences des *reads* longs corrigés. Cependant, les bases des

régions internes des *reads* longs n'ayant été couvertes par aucun contig et n'ayant pas pu être corrigées ne sont pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* longs corrigés ne peuvent pas être *splittés* après la correction.

3.2.3 Graphe de de Bruijn

Une autre alternative à l'alignement des *reads* courts sur les *reads* longs est l'utilisation directe d'un graphe de de Bruijn, construit à partir des *k*-mers des *reads* courts. Cette approche vise à s'affranchir de l'étape d'assemblage explicite des *reads* courts, utilisée par les méthodes présentées Section 3.2.2, et à directement utiliser le graphe afin de corriger les *reads* longs. Une fois le graphe construit, les *reads* longs peuvent en effet être ancrés sur ce dernier, et des chemins peuvent alors y être recherchés, afin de relier les régions ancrées, et ainsi corriger les régions non ancrées. Les méthodes présentées ici adoptent donc cette approche, et se différencient aussi bien par leur manière de représenter le graphe, que par les méthodologies utilisées afin d'ancrer les *reads* longs, et de corriger les régions non ancrées.

3.2.3.1 LoRDEC, 2014

LoRDEC commence par construire un graphe de de Bruijn à partir des *k*-mers solides des *reads* courts. Cette étape de filtration des *k*-mers faibles permet d'éviter d'introduire des erreurs lors de la correction des *reads* longs, en limitant la présence de bulles et de pointes dans le graphe. Les *k*-mers solides des *reads* longs sont ensuite utilisés comme points d'ancrage sur ce graphe de de Bruijn. Le graphe est alors parcouru, à partir des sommets correspondant à ces ancres, afin de trouver des chemins permettant de corriger les régions des *reads* longs composées uniquement de *k*-mers faibles. Ainsi, un *read* long ne disposant d'aucun *k*-mer solide ne peut pas être ancré sur le graphe, et ne peut donc pas être corrigé. Au vu des taux d'erreurs des *reads* longs, une petite taille de *k* doit alors être choisie pour la construction du graphe, afin de permettre la correction d'un maximum de *reads* longs. Ainsi, il est recommandé de construire le graphe pour des valeurs de *k* comprises entre 19 et 23. Deux approches de correction différentes sont ensuite mises en place, suivant si la région composée de *k*-mers faibles à corriger est bordée par des régions composées de *k*-mers solides, ou située sur une des extrémités d'un *read* long.

Dans le premier cas, les *k*-mers solides situés dans les régions bordantes de la région à corriger sont utilisés comme points d'ancrage sur le graphe. Les *k*-mers solides situés à gauche de la région à corriger sont qualifiés de *k*-mers *sources*, et les *k*-mers solides situés à droite sont qualifiés de *k*-mers *cibles*. Le graphe est alors traversé, afin de trouver un chemin entre un *k*-mer source et un *k*-mer cible, dictant une correction pour la région erronée. Afin de trouver un chemin optimal, à chaque embranchement, l'arc minimisant la distance d'édition avec la région erronée est choisi. Une limite sur le nombre maximal d'embranchements pouvant être explorés est cependant fixée, afin d'éviter les traversées trop coûteuses du graphe. Plusieurs paires de *k*-mers sont ainsi considérées, afin de limiter les cas où aucun chemin ne peut être trouvé entre deux *k*-mers. Le graphe étant traversé plusieurs fois pour la correction d'une même région, plusieurs de ces traversées peuvent cependant produire un chemin entre deux *k*-mers, et donc dicter différentes corrections pour la région erronée. Dans ce cas, le chemin dictant la séquence minimisant la distance d'édition à la région erronée du *read* long est alors choisi comme correction.

Dans le second cas, un des *k*-mers source ou cible est manquant, et la recherche de chemin ne peut alors pas simplement être arrêtée lorsqu'un chemin entre ces

deux *k*-mers est trouvé. Dans ce cas, le graphe est donc traversé uniquement à partir d'un *k*-mer solide proche de la région erronée. La traversée du graphe s'arrête alors lorsque la longueur de l'extrémité erronée sur *read* long est atteinte, que la distance d'édition entre le chemin actuel du graphe et l'extrémité erronée est trop importante, ou qu'aucun arc ne peut être suivi à partir du sommet courant. Une fois le parcours du graphe terminé, la séquence obtenue est alors réalignée sur l'extrémité du *read* long à corriger, afin de déterminer le préfixe (resp. le suffixe) de cette séquence s'alignant le mieux avec l'extrémité du *read* long. Le préfixe (resp. le suffixe) aligné de l'extrémité erronée du *read* long est alors corrigé avec le préfixe (resp. le suffixe) de la séquence. Cette étape d'alignement permet d'éviter d'utiliser comme correction des séquences trop divergentes avec l'extrémité du *read* long, causées par des traversées inappropriées du graphe.

Deux passes de correction sont ainsi appliquées pour chaque *read* long, une fois de la gauche vers la droite, et une fois de la droite vers la gauche. En effet, les *reads* longs étant corrigés à la volée par les traversées du graphe, ces corrections permettent de générer de nouveaux *k*-mers solides pouvant servir de point de départ à la deuxième passe de correction. De plus, les répétitions et les erreurs de séquençage présentes dans les *reads* courts peuvent amener la recherche de chemin à traverser différents sous-graphes, en fonction de l'extrémité de la région à corriger choisie comme point de départ.

LoRDEC produit ainsi des *reads* longs corrigés et non *trimmés*. Cependant, les bases correspondant à des *k*-mers solides sont reportées en majuscules, et les bases correspondant à des *k*-mers faibles sont reportées en minuscule. Ainsi, les *reads* longs corrigés peuvent facilement être *trimmés* ou *splittés* après la correction. Deux outils permettant de réaliser ces opérations sont fournis avec le programme.

3.2.3.2 Jabba, 2016

Contrairement aux méthodes présentées précédemment, Jabba commence par corriger les *reads* courts, afin de limiter l'impact des erreurs de séquençage sur la qualité du graphe de de Bruijn, et donc sur la qualité de la correction. Pour cela, il est recommandé d'utiliser Karect [1], bien que tout outil de correction puisse être utilisé. Le graphe de de Bruijn est alors construit à partir des *k*-mers des *reads* courts corrigés. Pour réduire davantage l'impact des erreurs présentes dans le graphe sur la correction des *reads* longs, une nouvelle procédure de correction est appliquée, cette fois sur le graphe lui-même. Plus précisément, les pointes sont éliminées, et les bulles sont retirées du graphe, en sélectionnant, pour chacune d'elle, uniquement le chemin le mieux supporté. Brownie [2] est utilisé pour construire et corriger le graphe.

De plus, contrairement à LoRDEC, Jabba ne repose pas sur l'étude des *k*-mers communs entre les sommets du graphe et les *reads* longs pour trouver des points d'ancrage, mais utilise plutôt des MEM entre les *reads* longs et les sommets du graphe. De ce fait, couplé à la haute qualité du graphe obtenu via les deux procédures de correction, de larges valeurs de *k* peuvent être utilisées pour sa construction. En pratique, le graphe est ainsi construit avec $k = 75$. Utiliser de telles valeurs de *k* permet de réduire la complexité du graphe, en résolvant les courtes répétitions, de taille inférieure à *k*, directement dans le graphe. Ainsi, l'alignement des *reads* longs sur le graphe lors de l'étape suivante est simplifié.

Une stratégie de type *seed and extend* est alors utilisée pour aligner les *reads* longs sur le graphe. Tout d'abord, les MEM, représentant les graines, sont calculés à l'aide d'essaMEM [114]. Cette méthode est basée sur une table des suffixes augmentée,

2. https://github.com/biointec/brownie

construite pour la séquence obtenue par concaténation de tous les sommets du graphe, et de leurs séquences complémentaires inverses. Une fois les graines d'un *read* long donné calculées, elles sont alors placées sur le graphe. Pour cela, plusieurs passes sont effectuées sur le *read* long. À chaque itération, toutes les régions du *read* long n'étant pas encore alignées sont considérées. Pour chacune de ces régions, les graines les plus longues sont utilisées, afin de déterminer les sommets sur lesquels cette région peut être alignée. La qualité des alignements entre chaque sommet et la région du *read* long est ensuite vérifiée. Les alignements inclus dans des alignements plus longs, et les alignements couvrant une trop petite fraction du sommet sont alors filtrés, et ne sont pas considérés lors de l'étape suivante.

Une fois les alignements locaux entre les régions du *read* long et les sommets du graphe calculés pour la passe courante, les alignements entre les différents sommets sont alors chaînés, en suivant des chemins du graphe. Chaque alignement local sur un sommet est ainsi étendu, en considérant tous les chemins possibles dans le graphe à partir de ce sommet. Si le sommet ne dispose que d'un seul arc sortant, alors l'alignement local est étendu le long de cet arc. Si plusieurs arcs sortants sont disponibles pour un sommet, la longueur des sommets cibles de chaque arc est étudiée. L'extension se faisant entre deux régions du *read* long, une distance maximale entre les deux sommets peut être définie, et les arcs trop longs ne sont alors pas considérés. Dans le cas où aucun arc sortant d'un sommet ne peut être suivi, la région correspondante du *read* long est alors reconsidérée et réalignée lors de la passe suivante.

Une fois toutes les passes effectuées, plusieurs alignements restent souvent possibles pour chaque *read* long. L'alignement couvrant la plus large proportion de la séquence du *read* long est alors sélectionné, et utilisé pour la correction. Pour la correction des extrémités du *read* long, l'alignement est étendu le long des chemins uniques du graphe. Si le *read* long s'étend en dehors de ces chemins uniques, les extrémités sont alors *trimmées*. En effet, corriger ces extrémités se révèle coûteux en temps, car elles doivent être alignées sur tous les chemins possibles, puisque ne possédant pas de graine pouvant guider l'alignement. Ce problème n'est donc pas abordé par Jabba, le gain de qualité étant trop faible par rapport au coût en temps.

Jabba produit ainsi des *reads* corrigés *splittés*. En effet, comme indiqué dans le paragraphe précédent, les extrémités des *reads* ne sont pas corrigées en dehors du plus long chemin unique du graphe sur lequel elles peuvent s'aligner. De plus, dans les cas où, même après plusieurs passes, aucun arc ne peut être suivi pour étendre l'alignement d'un sommet donné, un chemin direct de la première à la dernière graine d'un *read* long n'existe pas. Dans ce cas, Jabba *splitte* donc le *read* long, en corrigeant indépendamment les différentes régions de ce *read* ayant pu être alignées et chaînées le long du graphe.

3.2.3.3 FMLRC, 2018

FMLRC repose sur un principe similaire à celui de LoRDEC, à savoir construire un graphe de de Bruijn à partir des *reads* courts, et utiliser les k-mers solides des *reads* longs comme points d'ancrage sur ce graphe, afin de chercher des chemins permettant de corriger les régions des *reads* longs uniquement composées de k-mers faibles. Une des principales différences entre FMLRC et LoRDEC est que le graphe de de Bruijn est ici représenté implicitement, à l'aide d'un FM-index, construit pour la séquence obtenue en concaténant l'ensemble des *reads* courts. De ce fait, les arcs entre les différents sommets sont retrouvés en interrogeant le FM-index. La taille de k n'a donc pas besoin d'être connue au moment de la construction du graphe, mais

est directement choisie durant l'exécution. Ainsi, le FM-index permet de représenter implicitement tous les graphes de de Bruijn, et non un simple graphe d'ordre fixé. On parlera alors de graphe de de Bruijn *d'ordre variable*. Contrairement à LoRDEC, les *k*-mers faibles des *reads* court ne sont cependant pas filtrés, et tous les *k*-mers sont donc présents dans le graphe.

La correction des *reads* longs est ensuite réalisée de la même façon que dans LoRDEC. Pour les régions uniquement composées de *k*-mers faibles bordées par deux régions composées de *k*-mers solides, des *k*-mers sources et cibles sont définis sur chacune des régions bordantes et utilisés comme points d'ancrage sur le graphe. Un chemin est ensuite recherché dans le graphe, de la source vers la cible, et de la cible vers la source, la recherche pouvant parcourir différents sous-graphes en fonction du point de départ choisi. Comme pour LoRDEC, si plusieurs chemins sont trouvés, celui avec la plus faible distance d'édition avec la région erronée du *read* long est choisi comme correction. Pour les régions erronées situées aux extrémités des *reads* longs, le *k*-mer solide le plus proche de la région est utilisé comme point d'ancrage. Le graphe est alors traversé, la traversée s'arrêtant lorsque la longueur du chemin atteint la taille de la région erronée à corriger. Plusieurs chemins peuvent ainsi être obtenus, et celui minimisant la distance d'édition avec la région erronée du *read* long est de nouveau choisi comme correction.

Contrairement à LoRDEC, FMLRC réalise cependant deux passes de correction pour chaque *read* long. Une première passe est réalisée avec une petite valeur de *k* ($k = 21$ par défaut), et une seconde passe est ensuite réalisée avec une valeur de *k* plus large ($k = 59$ par défaut). La première passe suffit généralement à corriger la majorité de la séquence d'un *read* long, et la seconde passe permet alors de raffiner la correction, notamment en permettant la correction des régions répétées, plus aisées à résoudre en utilisant un graphe de de Bruijn d'ordre plus élevé. De plus, contrairement à LoRDEC, le seuil de solidité des *k*-mers des *reads* longs est ajusté dynamiquement pour chaque *read* long, en fonction de la taille de *k* choisie, et de la fréquence des *k*-mers du *read* long. La limite sur le nombre maximal d'embranchements pouvant être explorés est également fixée en fonction de la taille de *k*, afin de réduire le nombre d'embranchements explorés lors de la première passe. En effet, la traversée du premier graphe peut être coûteuse, au vu du grand nombre d'embranchements dans les régions répétées, dû à la faible taille de *k*. Ces régions sont alors résolues lors de la seconde passe de correction, en autorisant cette fois l'exploration d'un nombre d'embranchements plus important.

FMLRC produit ainsi des *reads* longs corrigés et non *trimmés*. Cependant, contrairement à LoRDEC, les bases correspondant à des *k*-mers faibles ne sont pas reportées dans une casse différente des bases correspondant à des *k*-mers solides. Ainsi, les *reads* longs corrigés ne peuvent pas être *trimmés* ou *splittés* après la correction.

3.2.4 Modèles de Markov cachés

Les modèles de Makov cachés, utilisés pour la correction de *reads* courts, ont également été adaptés à la correction des *reads* longs. Pour cela, les modèles sont tout d'abord initialisés afin de représenter les *reads* longs. Un sous-ensemble de *reads* courts est ensuite assigné à chaque *read* long, et est utilisé afin d'entraîner le modèle associé. Les modèles entrainés sont alors utilisés afin de déterminer des séquences consensus, et ainsi, corriger les *reads* longs qu'ils représentent.

3.2.4.1 Hercules, 2018

Comme pour LSC et LSCplus, Hercules commence par compresser les homopolymers des *reads* longs et des *reads* courts. Les *reads* courts compressés sont ensuite filtrés, en retirant ceux plus petits qu'une longueur fixée. Ces *reads* trop courts pourraient en effet s'aligner de façon ambiguë sur les *reads* longs. Les *reads* courts restants sont alors alignés sur les *reads* longs, par défaut à l'aide de Bowtie2, bien que tout aligneur proposant une sortie au format SAM puisse être utilisé. Une fois les alignements calculés, Hercules décompresse les *reads* longs et les *reads* courts, et recalcule les positions de début des alignements, en fonction des séquences décompressées. Cependant, contrairement aux autres méthodes se basant sur l'alignement de *reads* courts sur les *reads* longs, Hercules n'utilise par la suite que la position de début d'alignement, et aucune des autres informations fournies par l'aligneur. De ce fait, la détermination des bases correctes à utiliser pour corriger les *reads* longs n'est ici pas réalisée à l'aide de l'aligneur.

Chaque *read* long est ensuite modélisé à l'aide d'un *profile Hidden Markov Model* (*pHMM*). Ces pHMM disposent de trois types d'états, permettant de représenter des délétions, des insertions, ou des *matches* / substitutions. Le but d'Hercules est alors de produire, à partir du pHMM d'un *read* long, sa séquence consensus, définie comme la séquence la plus probable, parmi toutes les séquences que le modèle peut produire. Cependant, le consensus généré par un pHMM traditionnel est uniquement basé sur les états de *matches*. Ainsi, afin de pouvoir également prendre en compte les états d'insertions et de délétions dans le calcul du consensus, les pHMM sont modifiés. Les boucles sur les états d'insertions sont ainsi retirées, et remplacées par plusieurs états d'insertions distincts. Les états de délétions sont quant à eux retirés, et remplacés par des transitions de délétions. Une fois le modèle construit, les probabilités d'émission et de transmission de chaque état sont initialisées à l'aide du profil d'erreur de la technologie de séquençage utilisée.

Le modèle est ensuite mis à jour, à l'aide des informations des *reads* courts. Pour cela, chaque *read* court aligné sur un *read* long est considéré indépendamment. Le sous-graphe correspondant à la région du *read* long couverte par ce *read* court est alors extrait. Ce sous-graphe est ensuite entraîné, à l'aide des informations contenues dans le *read* court sélectionné, en utilisant l'algorithme *Forward-Backward* [8]. Les probabilités d'émission et de transmission de chaque sommet sont ainsi mises à jour par l'algorithme, indépendamment et exclusivement dans chaque sous-graphe, en utilisant les probabilités du modèle initial. Dans le cas où un sommet est considéré et mis à jour dans différents sous-graphes, ses probabilités finales, dans le modèle entraîné, sont alors définies en calculant la moyenne de ses probabilités, dans chaque sous-graphe dans lequel il apparaît. Les régions non couvertes des *reads* long ne pouvant produire de sous-graphes à entraîner, les probabilités d'émission et de transmission du modèle initial sont conservées, pour les sommets correspondants. Une fois tous les *reads* courts alignés considérés, et tous les sous-graphes entraînés, le modèle est alors mis à jour à partir de ces sous-graphes. L'algorithme de Viterbi [112] est ensuite utilisé, afin de décoder la séquence consensus du modèle mis à jour. Cet algorithme considère le pHMM de chaque *read* long, et recherche un chemin, de l'état initial à l'état final, produisant les probabilités d'émission et de transmission les plus élevées, via une approche par programmation dynamique.

Hercules produit ainsi des *reads* longs corrigés et non *trimmés*. En effet, la séquence consensus d'un *read* long étant extraite du pHMM en recherchant un chemin du premier au dernier état, représentant respectivement la première et la dernière base de ce *read*, la totalité de la longueur des *reads* longs originaux est préservée par

la correction. De plus, les bases provenant de sous-graphes ayant pu être entraînés ne sont pas reportées dans une casse différente des bases non couvertes par des *reads* courts. Ainsi, les *reads* longs corrigés ne peuvent pas être *trimmés* ou *splittés* après la correction.

3.3 Auto-correction

L'auto-correction vise à s'affranchir de la nécessité de *reads* courts complémentaires, et à corriger les *reads* longs en se basant uniquement sur les informations que contiennent leurs séquences. L'auto-correction se divise en deux principales approches.

1. L'alignement multiple. Cette approche est similaire aux approches de correction hybride par alignement de *reads* courts ou de contigs, décrites Section 3.2.1 et Section 3.2.2. En effet, une fois alignés entre eux, les *reads* longs peuvent être corrigés, en calculant, pour chacun d'eux, une séquence consensus à partir du sous-ensemble de *reads* longs alignés. PBDAGCon (le module de correction de l'assembleur HGAP) [19], PBcR-BLASR [55], Sprai [3] [35], PBcR-MHAP [9], FalconSense (le module de correction de l'assembleur Falcon) [18], Sparc [127], le module de correction intégré dans l'assembleur Canu [57], MECAT [124] et FLAS [7] reposent sur cette approche.
2. L'utilisation d'un graphe de de Bruijn. D'une manière similaire à l'approche de correction hybride par graphe de de Bruijn décrite Section 3.2.3, une fois le graphe construit, les *reads* longs peuvent en effet y être ancrés. Le graphe peut alors être traversé, afin de trouver des chemins permettant de relier les régions ancrées des *reads* longs, et ainsi corriger les régions non ancrées. LoRMA [100] et Daccord [110] se basent sur cette approche.

Pour alléger les notations, l'ambiguïté n'étant plus possible, puisque les méthodes d'auto-correction se passent de *reads* courts, le terme *reads* désignera, dans cette section, les *reads* longs.

3.3.1 Alignement multiple

Cette approche est similaire à la correction hybride par alignement de *reads* courts décrite Section 3.2.1, et à la correction hybride par alignement de contigs décrite Section 3.2.2. Elle consiste ainsi en une première étape d'alignement des *reads* entre eux, afin de déterminer, pour chaque *read*, un ensemble de *reads* similaires. Dans un second temps, cet ensemble de *reads* est alors utilisé afin de calculer une séquence consensus de haute qualité, et de corriger le *read* original. La principale différence avec les approches hybrides provient des méthodes d'alignement utilisées, ces dernières variant en fonction des propriétés des *reads* auxquels elles s'appliquent, comme mentionné Section 2.5.1. Les choix algorithmiques réalisés afin de calculer les séquences consensus diffèrent également, aussi bien par rapport aux méthodes de correction hybride, qu'entre les différentes méthodes d'auto-correction adoptant cette approche.

3.3.1.1 PBDAGCon, 2013

PBDAGCon est un module de correction, inclus dans l'assembleur HGAP. Pour commencer, les alignements entre les *reads* sont calculés à l'aide de BLASR. Pour

3. http://zombie.cb.k.u-tokyo.ac.jp/sprai/index.html

chaque *read*, un DAG est ensuite construit, à partir de l'ensemble de ses *reads* similaires, déterminé par l'étape d'alignement précédente. Les sommets de ce graphe sont étiquetés en fonction des bases des *reads*, et un arc est présent entre deux sommets s'il existe un *read* contenant consécutivement les deux bases correspondantes à ces sommets. Ainsi, chaque *read* peut être représenté par un chemin unique dans le graphe. Chaque sommet du graphe possède donc un ensemble de *reads* le supportant, et chaque arc possède également un ensemble de *reads* le traversant. Le graphe est donc pondéré, en définissant le poids d'un arc comme le nombre de *reads* le traversant, *i.e.*, le nombre de *reads* possédant consécutivement les deux bases correspondantes aux sommets reliés par cet arc.

Le graphe est construit en l'initialisant tout d'abord avec la séquence du *read* initial à corriger. Tous les arcs sont ainsi initialisés avec un poids de 1. Le graphe est ensuite mis à jour, en y ajoutant séquentiellement les autres *reads*, à partir des informations des alignements. Ainsi, si un alignement correspond à des sommets et à des arcs déjà présents dans le graphe, les poids de ces arcs sont alors incrémentés. Dans le cas contraire, de nouveaux sommets et de nouveaux arcs sont créés, en initialisant les poids de ces nouveaux arcs à 1.

Une fois tous les *reads* ajoutés, et le graphe final obtenu, la séquence consensus du *read* initial peut être générée. Pour cela, un score est assigné à chaque sommet, en fonction des poids de l'ensemble de ses arcs sortants. Le chemin maximisant la somme des scores des sommets traversés est alors recherché dans le graphe, à l'aide d'un algorithme de programmation dynamique. Ce chemin est recherché en prenant comme point de départ le sommet du graphe représentant la première base du *read* initial, et comme point d'arrivée le sommet du graphe représentant sa dernière base.

PBDAGCon produit ainsi des *reads* corrigés et non *trimmés*. En effet, la séquence consensus d'un *read* est calculée en recherchant un chemin dans le graphe entre les sommets représentant la première et la dernière base de ce *read*. La totalité de la longueur des *reads* originaux est ainsi préservée par la correction. De plus, les bases provenant des *reads* originaux et n'ayant pas pu être corrigées, à cause d'un manque de couverture, ne sont pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* corrigés ne peuvent pas être *trimmées* ou *splittés* après la correction.

3.3.1.2 PBcR-BLASR, 2013

PBcR-BLASR est une version modifiée de la méthode de correction hybride PBcR, présentée Section 3.2.1, et permettant une adaptation à l'auto-correction. Ainsi, les alignements entre les *reads* sont tout d'abord calculés à l'aide de BLASR. Les alignements ainsi obtenus sont ensuite utilisés afin de corriger les *reads*, à l'aide de l'algorithme de correction de PBcR.

PBcR-BLASR utilisant l'algorithme de PBcR pour la correction, les *reads* produits sont ainsi *splittés*, pour les raisons mentionnées dans la version hybride de PBcR, Section 3.2.1.

3.3.1.3 Sprai, 2014

Sprai commence par calculer les alignements entre les *reads* à l'aide de BLAST+. L'ensemble des *reads* similaires s'alignant avec un *read* donné peut alors être utilisé afin de définir un alignement multiple. Cet alignement multiple est ensuite raffiné, à l'aide de ReAligner [3], afin de maximiser son score en modifiant son agencement, tout en conservant sa structure globale. Le but recherché est alors de regrouper les bases similaires sur des colonnes identiques de la matrice représentant l'alignement

multiple. En effet, des choix locaux sont effectués lors de l'alignement de deux *reads*, en particulier au niveau des positions des insertions et des délétions. Ces dernières étant fréquentes dans les *reads* longs, la combinaison des alignements associés à un *read* en un alignement multiple n'assure donc pas la cohérence des choix réalisés au sein de tous les alignements. Le raffinement vise alors à rétablir cette cohérence, en apportant des modifications locales à l'alignement multiple, afin de réunir davantage de support à chaque position.

Pour cela, toutes les séquences incluses dans l'alignement multiple sont considérées indépendamment. Chacune de ces séquences est alors réalignée sur l'alignement multiple, dont la séquence en cours de traitement a été exclue, par programmation dynamique. L'alignement multiple est alors mis à jour, en fonction du résultat d'alignement obtenu. Les autres séquences sont ensuite considérées, et le processus est ainsi répété, jusqu'à ce que le score de l'alignement multiple cesse d'augmenter. L'alignement multiple final ainsi obtenu est alors utilisé afin de calculer la séquence consensus du *read* original, par vote majoritaire à chaque position.

Sprai produit ainsi des *reads* corrigés non *trimmés*. En effet, l'alignement multiple est ici construit à l'échelle du *read* complet. De ce fait, les consensus extraits par vote majoritaire contiennent également les extrémités non couvertes des *reads*. De plus, les bases provenant des *reads* originaux et n'ayant pas pu être corrigées, à cause d'un manque de couverture, ne sont pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* corrigés ne peuvent pas être *trimmés* ou *splittés* après la correction.

3.3.1.4 PBcR-MHAP, 2015

PBcR-MHAP est une version améliorée de PBcR-BLASR. Dans cette version, le calcul des alignements entre les *reads*, préalablement réalisé à l'aide de BLASR, est notamment remplacé par une approche par *mapping*, appelée MHAP (*MinHash Alignment Process)*. Dans cette approche, les *k*-mers des *reads* sont tout d'abord extraits, et convertis en empreintes entières à l'aide de fonctions de hachage. Un *sketch*, dont le nombre d'entrées est défini par le nombre de fonctions de hachage utilisées, est ainsi construit pour chaque *read*. Pour chacune des fonctions de hachage, le *k*-mer du *read* générant la plus petite valeur entière, est choisi pour faire partie du *sketch*. Ce *k*-mer est alors appelé *min-mer*. Le *sketch* d'un *read* est ainsi défini comme l'ensemble ordonné des empreintes entières de ses *min-mers*. La fraction d'entrées communes entre les *sketches* de deux *reads* peut alors être utilisée pour estimer leur similarité. Ainsi, si la similarité entre deux *sketches* est suffisante, les *min-mers* communs sont localisés dans les *reads* correspondants, et la différence médiane entre leurs positions est calculée afin de déterminer les positions du chevauchement entre les deux *reads*. Une deuxième passe de ce procédé est alors appliquée, cette fois uniquement sur les régions chevauchantes, afin d'obtenir une meilleure estimation de la similarité entre ces séquences.

Une fois les chevauchements entre tous les *reads* calculés, ces derniers peuvent alors être corrigés à partir de ces informations. L'algorithme utilisé dans PBcR et PBcR-BLASR est cependant abandonné au profit de deux modules de consensus : FalconSense, plus rapide, mais moins robuste, et PBDAGCon, plus lent mais plus précis. Ces deux algorithmes ne sont pas présentés ici, mais sont détaillés, respectivement, Section 3.3.1.5 et Section 3.3.1.1. De plus, ces deux algorithmes ne sont pas appliqués l'un à la suite de l'autre, et le choix de l'algorithme à utiliser est alors laissé à l'utilisateur.

PBcR-MHAP produit ainsi des *reads* corrigés *splittés*, si FalconSense est utilisé, et des *reads* corrigés non *trimmés*, dont les bases non corrigées ne sont pas reportées dans une casse différente des bases corrigées, si PBDAGCon est utilisé. Les détails concernant les spécificités de ces *reads* corrigés sont donnés dans les descriptions respectives de ces deux algorithmes, Section 3.3.1.5 et Section 3.3.1.1.

3.3.1.5 FalconSense, 2016

FalconSense est un module de correction, inclus dans l'assembleur FALCON. Pour commencer, les alignements entre les *reads* sont calculés à l'aide de DALIGNER. L'ensemble des *reads* similaires s'alignant avec un *read* donné est ensuite indépendamment réaligné sur ce dernier, en n'autorisant cependant pas les substitutions, et en encodant donc toutes les opérations d'édition par des insertions et des délétions. Pour chaque position d'un alignement ainsi obtenu entre le *read* initial A et un de ses *reads* similaires B, une étiquette est générée. Chacune de ces étiquettes est composée des quatre champs suivants :

1. La position correspondante sur le *read* A.
2. Le décalage par rapport à cette position sur le *read* A. Cette valeur est différente de 0 lorsqu'une insertion est présente dans le *read* B à cette position de l'alignement, et indique ainsi la taille du décalage entre la position sur le *read* A, et la position sur l'alignement.
3. La base de A à cette position de l'alignement.
4. La base de B à cette position de l'alignement.

Une fois toutes les étiquettes définies, pour toutes les positions de tous les alignements entre le *read* initial et ses *reads* similaires, un graphe d'alignement est construit. Ce graphe est un DAG pondéré, dont les sommets sont définis par les étiquettes. Un arc est créé entre deux étiquettes si ces dernières sont consécutives dans un des alignements. Le poids de cet arc est alors initialisé à 1 s'il n'était pas déjà présent dans le graphe, et dans le cas contraire, le poids de l'arc déjà présent est incrémenté. Une fois le graphe final construit, l'étiquette associée à un arc entre deux sommets représente alors le nombre de *reads* supportant la connexion entre ces deux sommets.

La séquence consensus du *read* initial peut ensuite être déterminée, en appliquant un algorithme de programmation dynamique sur le graphe, afin de trouver le chemin de poids le plus élevé entre les sommets correspondant à la première et la dernière étiquette associées au *read* initial. Les bases associées au quatrième champs des étiquettes situées le long de ce chemin dictent alors la séquence consensus.

FalconSense produit ainsi des *reads* corrigés *splittés*. En effet, la séquence consensus d'un *read* est calculée en recherchant un chemin dans le graphe entre les sommets correspondant à la première et la dernière étiquette associées à ce *read*. Ces étiquettes étant calculées en fonction des alignements de ce *read*, ses extrémités non couvertes impliquent un manque d'étiquettes, et donc un manque de sommets et d'arcs dans le graphe. De ce fait, ces extrémités non couvertes ne peuvent pas être incluses dans le calcul de la séquence consensus, et ne sont donc pas reportées. De plus, les régions internes d'un *read*, non couvertes par des alignements, impliquent également un manque d'étiquettes, et donc un manque de sommets et d'arcs dans le graphe. Dans ce cas, différents parcours sont alors effectués sur les différents sous-graphes correspondant aux régions couvertes de ce *read*. Différentes séquences consensus sont ainsi calculées indépendamment pour chacune de ces régions.

3.3.1.6 Sparc, 2016

Sparc commence par aligner les *reads* les uns avec les autres, à l'aide de BLASR. Pour chaque *read*, un graphe de de Bruijn modifié est ensuite construit, à partir de l'ensemble de ses *reads* similaires, déterminé par l'étape d'alignement précédente. Contrairement à un graphe de de Bruijn classique, les sommets de ce graphe sont définis non seulement en fonction des *k*-mers des *reads*, mais également en fonction des positions de ces *k*-mers dans les *reads*. Deux sommets indépendants sont donc construits pour deux *k*-mers identiques, s'ils sont situés à des positions différentes, contrairement à un graphe de de Bruijn classique, ne définissant qu'un unique sommet dans ce cas. Cette nécessité de créer différents sommets pour un même *k*-mer situé à des positions différentes s'explique notamment par les faibles valeurs de *k* utilisées par Sparc ($k \leq 3$ en pratique). De plus, représenter les *k*-mers identiques à des positions différentes par des sommets distincts permet également d'éviter les cycles au sein du graphe, contraignant ainsi ce dernier à être un DAG. Bien que les valeurs de *k* utilisées soient faibles, ce graphe est cependant différent du DAG utilisé dans PBDAGCon, ce dernier ne stockant non pas de petits *k*-mers, mais simplement les bases des *reads*.

De plus, le graphe construit par Sparc est échantillonné, en ne stockant qu'un *k*-mer toutes les *n* bases. Les arcs entre les sommets représentent alors les *k*-mers consécutifs des *reads*, et sont étiquetés par les séquences permettant de relier les sommets associés. Le graphe est également pondéré, en associant un poids à chaque arc, représentant le nombre de *reads* supportant cet arc. Le graphe est ainsi initialisé à partir du *read* initial, en associant un poids de 1 à tous les arcs, et est ensuite mis à jour en fonction des alignements associés à ce *read*, à la manière de PBDAGCon. Les poids des arcs sont ainsi incrémentés si un alignement donné correspond à une région existante du graphe, et de nouveaux sommets et de nouveaux arcs sont créés dans le cas contraire. Une fois le graphe final obtenu, un consensus peut alors être extrait, en recherchant le chemin de poids maximal, du premier au dernier sommet associés au *read* initial, à l'aide d'un algorithme de programmation dynamique, à la manière de PBDAGCon.

Sparc produit ainsi des *reads* corrigés et non *trimmés*. En effet, la séquence consensus d'un *read* est obtenue en recherchant un chemin du premier au dernier sommet du graphe associés à ce *read*. Ces sommets, représentant respectivement le premier et le dernier *k*-mer de ce *read*, la totalité de la longueur des *reads* originaux est alors préservée par la correction. Comme pour PBDAGCon, les bases provenant des *reads* originaux et n'ayant pas pu être corrigées, à cause d'un manque de couverture, ne sont pas reportées dans une casse différente des bases corrigées. Ainsi, les *reads* corrigés ne peuvent pas être *trimmés* ou *splittés* après la correction.

De plus, Sparc dispose également d'un mode hybride, permettant d'utiliser des *reads* courts en complément lors de l'alignement. Dans ce cas, le processus de correction reste globalement inchangé, la seule différence étant que des poids plus importants sont accordés aux arcs supportés par des *reads* courts, afin de favoriser le choix de ces arcs lors du calcul de la séquence consensus.

3.3.1.7 Canu, 2017

Le module de correction de l'assembleur Canu est une version améliorée de PBcR-MHAP, présenté Section 3.3.1.4. Pour l'étape de *mapping*, la création des *sketches* des *reads* est modifiée, afin de permettre d'ajuster la probabilité d'inclure

ou non certains *k*-mers. En effet, un *k*-mer apparaissant régulièrement dans l'ensemble des *reads* devrait avoir un poids plus faible, puisque peu représentatif de l'origine réelle d'un *read* dans le génome. Au contraire, un *k*-mer présent dans peu de *reads*, mais apparaissant plusieurs fois dans un seul et même *read*, devrait avoir un poids plus élevé, puisque représentant une proportion plus importante de la longueur de ce *read*. Une combinaison de ces poids, appelée poids *tf-idf* (*term frequency, inverse document frequency*) est ainsi appliquée à chacun des *k*-mers.

Pour cela, pour chacune des entrées du *sketch* d'un *read*, plutôt que d'appliquer une unique fonction de hachage à chaque *k*-mer, un nombre de fonctions de hachage, correspondant au poids *tf-idf* du *k*-mer en question est appliqué. Ainsi, les *k*-mers disposant d'un poids important seront hachés plusieurs fois, augmentant leurs chances de faire partie du *sketch*. Cette application du poids *tf-idf* aux *sketches* permet d'augmenter la sensibilité des chevauchements, tout en diminuant le nombre de chevauchements répétitifs. La deuxième phase du procédé, permettant de mieux estimer la similarité entre les séquences chevauchantes, et également modifiée par rapport à PBcR-MHAP, en utilisant une unique fonction de hachage pour construire les *sketches* des régions chevauchantes.

Comme dans PBcR-MHAP, les chevauchements ainsi calculés sont alors utilisés pour corriger pour les *reads*. Cette étape est également améliorée, en permettant notamment de sélectionner un sous-ensemble optimal de chevauchements à utiliser pour la correction de chaque *read*, via une stratégie similaire à la version hybride de PBcR. Les alignements base à base n'étant cependant pas disponibles, le score associé à chaque *chevauchement* est calculé en fonction de sa longueur, et de l'estimation de la similarité des deux séquences. Chaque *read* est ainsi uniquement autorisé à participer à la correction de ses *N reads* les plus similaires, *N* représentant la profondeur de séquençage des *reads*. Cette approche permet notamment de réduire les biais induits par les régions répétées, en évitant que ces régions soient toujours couvertes par les mêmes ensembles de *reads*.

Les chevauchements conservés pour chaque *read* sont ensuite utilisés pour la correction. Celle-ci est réalisée à l'aide du module de consensus FalconSense, ayant été choisi comme algorithme par défaut, et le support de PBDAGCon ayant été abandonné. De plus, l'algorithme de FalconSense est légèrement modifié, en retirant du graphe les arcs dont les poids sont trop faibles, afin de s'assurer que le calcul de la séquence consensus ne traverse que des arcs suffisamment supportés.

Le module de correction de Canu produit ainsi des *reads* corrigés *splittés*, pour les raisons mentionnées dans la description de FalconSense, Section 3.3.1.5.

3.3.1.8 MECAT, 2017

MECAT utilise également une stratégie de *mapping* pour calculer les chevauchements entre les *reads*. Cette stratégie est basée sur la détermination de *k*-mers communs entre les *reads*, afin de définir les chevauchements. Pour cela, les *reads* sont tout d'abord découpés en plusieurs blocs, de 1 000 à 2 000 paires de bases. Les *reads* sont ainsi indexés dans une table de hachage, en utilisant leurs *k*-mers comme clés, et les positions de ces *k*-mers au sein des différents blocs comme valeurs. Pour trouver des chevauchements entre les *reads*, les *k*-mers des blocs de ces *reads* sont alors parcourus et recherchés dans la table de hachage. Un bloc d'un *read* est ainsi défini comme chevauchant un bloc d'un autre *read* si ces deux blocs partagent un nombre suffisant de *k*-mers. Par extension, un *read* est défini comme chevauchant un autre *read* si au moins une paire de blocs de ces *reads* se chevauche.

Une étape de filtration des chevauchements excessifs et non informatifs est ensuite appliquée, via un système de score. Pour cela, des paires de *k*-mers sont considérées au sein de deux blocs chevauchants. Une distance, appelée *DDF* (*Distance Difference Factor*), est alors calculée entre les paires de *k*-mers communs entre les deux blocs, en fonction des positions de leurs occurrences respectives au sein de ces blocs. Si cette distance est plus faible qu'un seuil donné, les *k*-mers considérés se supportent mutuellement, et leurs scores sont alors incrémentés. Le *k*-mer possédant le plus haut score d'un bloc est ensuite défini comme une graine pour l'étape suivante, si ce score est assez élevé. Si plusieurs *k*-mers partagent ce score, l'un d'eux est alors choisi aléatoirement.

Le système de score est ensuite étendu du bloc courant aux blocs voisins, après obtention de la graine. Les DDF sont alors calculées entre cette graine et les *k*-mers des blocs voisins. De la même façon que précédemment, si la DDF d'une paire de *k*-mers est plus faible qu'un seuil donné, le score de la graine est incrémenté. Si plus de 80% des DDF d'un bloc voisin sont plus faibles que le seuil, le bloc est alors marqué, est les scores des *k*-mers de ce bloc ne sont pas calculés. Si des blocs restent non marqués après une étape de calcul de scores, le processus est alors répété sur ces blocs, en calculant les scores de leurs *k*-mers, comme indiqué dans le paragraphe précédent.

Une fois tous les scores des *k*-mers communs entre deux *reads* chevauchants calculés, un alignement base à base peut alors être calculé entre ces derniers. Pour cela, les *k*-mers de ces *reads* sont triés en fonction de leurs scores. Les *k*-mers possédant les scores les plus élevés sont alors utilisés en tant que graines, afin de faciliter le calcul de l'alignement. Si la longueur de l'alignement final est au moins aussi grande que la taille d'un bloc, et que la divergence entre les deux séquences est suffisamment faible, l'alignement est alors considéré comme correct, et est reporté.

Pour corriger un *read* donné, les *reads* le chevauchant sont triés, et considérés dans l'ordre décroissant des scores cumulés de leurs *k*-mers, calculés à l'étape précédente via les DDF. Les alignements locaux base à base sont ainsi calculés entre le *read* à corriger, et les *reads* le chevauchant, dans l'ordre décroissant de leurs scores. De plus, afin d'éliminer les biais liés aux *reads* chimériques et aux séquences répétées, un alignement entre deux *reads* est filtré si sa longueur est plus faible que 90% de la taille du *read* le plus court. Les calculs des alignements locaux sont alors arrêtés dès lors que 100 alignements ont été produits, ou que tous les *reads* ont été considérés. Le score DDF permettant d'estimer la longueur des chevauchements, calculer les alignement locaux uniquement entre le *read* à corriger et ses *reads* chevauchants ayant les scores les plus élevés permet de récupérer suffisamment d'information pour permettre la correction rapidement, tout en évitant de calculer des alignements répétitifs, n'apportant que peu d'information supplémentaire.

Une nouvelle stratégie de correction, combinant les principes de PBDAGCon et de FalconSense est ensuite utilisée. Pour cela, les alignements du *read* à corriger sont résumés dans une table de consensus, contenant les comptes de *matches*, d'insertions, et de délétions, pour chaque position du *read*. Cette table est alors parcourue, afin de déterminer trois types de régions, nécessitant des approches de correction différentes.

1. Les régions triviales, composées de *matches* consistants, et d'un faible nombre d'insertions (typiquement, < 6).
2. Les régions composées de délétions consistantes, et d'un faible nombre d'insertions (< 6).

3. Les régions plus complexes, et comprenant de plus nombreuses insertions ($\geq$ 6).

Les deux premiers types de régions sont corrigées en déterminant simplement la base consensus en fonction des comptes contenus dans la table. Pour le troisième type de régions, un DAG est construit, et le consensus est alors déterminé en recherchant le chemin le mieux supporté de ce DAG, via une approche de programmation dynamique. Ces régions étant généralement courtes ($<$ 10 bases), le consensus peut rapidement être calculé à partir du DAG.

MECAT produit ainsi des *reads* corrigés *splittés*. En effet, les blocs des *reads* pour lesquels aucun chevauchement n'a pu être déterminé, qu'ils soient situés aux extrémités des *reads* ou non, ne peuvent être corrigées. Ces derniers sont donc éliminés des séquences des *reads* corrigés, et les blocs consécutifs possédant des chevauchements sont ainsi reportés séparément.

3.3.1.9 FLAS, 2019

FLAS est une surcouche de MECAT, présenté précédemment. Il permet de raffiner les chevauchements produits avant la correction, et d'utiliser les régions corrigées des *reads* afin d'en corriger les régions non corrigées. Pour commencer, les chevauchements entre les *reads* sont donc calculés à l'aide de la stratégie de *mapping* de MECAT. Un graphe dirigé est alors construit à partir de ces chevauchements. Chaque sommet de ce graphe représente un *read*, et un arc est présent entre deux sommets u et v si les *reads* associés à ces sommets possèdent au moins un chevauchement de longueur suffisante. Une fois le graphe construit, ses cliques maximales sont alors recherchées, à l'aide de l'algorithme de Bron-Kerbosch [12, 24, 25].

Les paires de cliques maximales partageant des sommets communs sont alors considérées, afin de trouver des chevauchements additionnels entre les *reads* correspondant, ou de retirer des chevauchements erronés. Trois cas différents peuvent en effet être à l'origine des sommets communs dans une paire de cliques maximales donnée. Ces sommets communs nécessitent alors des traitements adaptés.

1. Les *reads* des deux cliques proviennent de la même région du génome, et les chevauchements entre les *reads* des sommets communs sont donc corrects.
2. Les *reads* des deux cliques proviennent de régions différentes du génome, mais les *reads* des sommets communs traversent ces deux régions, et leurs chevauchements sont donc corrects.
3. Les *reads* des deux cliques proviennent de régions différentes du génome, et les *reads* des sommets communs proviennent de l'une de ces régions, et partagent des chevauchements erronées avec les *reads* provenant de l'autre région.

Ces différents cas peuvent être déterminés en comparant le nombre de sommets communs s au nombre de sommets de la plus petite clique maximale de la paire c. En effet, parmi tous les chevauchements, le nombre de chevauchements corrects est généralement plus élevé que le nombre de chevauchements erronés. Ainsi, si s représente moins de 50% des sommets de c, les sommets communs sont considérés comme faisant partie du troisième cas.

Dans le cas contraire, les sommets communs sont alors considérés comme faisant partie du premier ou du deuxième cas. Pour distinguer ces deux cas, le nombre de *reads* des sommets communs ayant une région chevauchant les *reads* d'une des deux cliques, et une région différente chevauchant les *reads* de l'autre clique, est calculé. Si ce nombre représente au moins 50% de l'ensemble des sommets communs aux deux

cliques, ces derniers sont alors considérés comme faisant partie du deuxième cas. Dans le cas contraire, les sommets communs sont donc considérés comme faisant partie du premier cas.

Pour les cliques maximales faisant partie du premier cas, les chevauchements entre les *reads* correspondant aux sommets des cliques sont recalculés, afin de produire des chevauchements supplémentaires, et de pouvoir ainsi corriger de plus larges régions de ces *reads*.

Pour les cliques maximales faisant partie du deuxième cas, les *reads* des sommets communs traversant simplement les deux régions du génome correspondant aux cliques, aucun chevauchement supplémentaire ne peut être calculé, et aucune action n'est donc nécessaire.

Enfin, pour les cliques maximales faisant partie du troisième cas, les sommets communs doivent être assignés à une seule des deux cliques. Pour cela, chaque sommet commun est considéré indépendamment. La similarité moyenne des chevauchements entre le *read* associé au sommet et les autres *reads* de chaque clique est alors calculé. Le sommet est ainsi retiré de la clique affichant la similarité moyenne la plus faible.

Une fois les chevauchements ainsi raffinés, en fonction des trois différents cas, l'algorithme de MECAT est alors utilisé afin de corriger les *reads*.

Une seconde étape de correction est ensuite appliquée, et les régions corrigées des *reads* sont utilisées afin de corriger les régions n'ayant pas pu être corrigées par la première étape. Pour cela, les chevauchements entre les régions corrigées des *reads* sont calculés à l'aide de la stratégie de *mapping* de MECAT. Un graphe représentant les chevauchements est alors construit, comme lors de l'étape précédente. Les *reads* sont ensuite alignés sur ce graphe, afin de procéder à la correction. Pour les *reads* comportant des régions corrigées et des régions non corrigées, les régions corrigées peuvent être alignées de façon unique sur le graphe, et servir de points d'ancrage. Les régions non corrigées peuvent alors être alignées le long des chemins reliant ces régions corrigées, et donc être corrigées à l'aide des informations contenues dans les chemins suivis. De plus, les *reads* uniquement composés de régions non corrigées peuvent également être alignés sur des chemins du graphe, au vu du faible nombre d'erreurs présentes dans ce dernier. Dans le cas où une région non corrigée d'un *read* s'aligne sur plusieurs chemins, le chemin sur lequel la région s'aligne avec la plus haute identité est choisi comme correction. Si aucun alignement ne se démarque, le chemin le plus largement couvert par les *reads* est alors choisi.

FLAS produit ainsi des *reads* corrigés *splittés*. En effet, FLAS repose sur les processus de *mapping* et de correction de MECAT. Les régions (ou les blocs, comme défini dans MECAT) des *reads* pour lesquels aucun chevauchement n'est déterminé ne peuvent donc pas être corrigées. Ces dernières sont donc éliminées des séquences des *reads* corrigés, qu'elles soient situées aux extrémités de ces *reads* ou non. Les régions consécutives possédant des chevauchements sont ainsi reportées séparément. Cependant, contrairement à MECAT, FLAS produit également les *reads* corrigés en version *trimmée*, conservant alors les régions internes n'ayant pas pu être corrigées à cause d'un manque de chevauchements.

3.3.2 Graphe de de Bruijn

Cette approche est similaire à la correction hybride par graphe de de Bruijn, décrite Section 3.2.3. Elle consiste en une première étape de construction du graphe, et en une seconde étape d'ancrage des *reads* sur le graphe, et de recherche de chemins permettant de relier les régions ancrées, afin de corriger les régions non ancrées. La

principale différence avec l'approche hybride provient du fait que le graphe n'est ici construit qu'à partir des *k*-mers contenus dans les *reads* longs. Les méthodes présentées ici adoptent donc cette approche, et diffèrent principalement par l'échelle à laquelle le graphe est construit. En effet, le graphe peut être construit globalement, en étudiant la fréquence de l'ensemble des *k*-mers des *reads*, ou localement, en alignant au préalable les *reads* entre eux, et en définissant et construisant le graphe uniquement sur de petites régions similaires des *reads*.

3.3.2.1 LoRMA, 2016

LoRMA reprend le principe proposé par LoRDEC pour la correction hybride, en l'adaptant à l'auto-correction. Le graphe est ainsi construit à partir des *k*-mers solides des *reads*. Ces derniers peuvent alors être ancrés sur le graphe, en fonction des *k*-mers solides qu'ils possèdent. De la même façon que LoRDEC, les régions composées de *k*-mers faibles, bordées par des régions composées de *k*-mers solides, sont alors corrigées en recherchant, dans le graphe, des chemins permettant de relier une ancre de la région de gauche à une ancre de la région de droite. De manière similaire, les régions composées de *k*-mers faibles, et situées aux extrémités des *reads*, sont corrigées en utilisant un *k*-mer solide voisin comme ancre, et en traversant le graphe, en suivant les mêmes conditions d'arrêt que pour LoRDEC.

Le principe de FMLRC, proposant plusieurs passes de correction, avec des valeurs croissantes de *k*, est également repris. Au vu des taux d'erreurs initiaux des *reads*, la première passe de correction est ainsi réalisée avec une petite valeur de *k*. Bien que de petites valeurs de *k* ne permettent pas de corriger de larges régions de *k*-mers faibles, dû au nombreux embranchements présents dans le graphe, de petites régions peuvent tout de même être corrigées. De ce fait, après chaque passe de correction, les régions de *k*-mers solides sont alors plus longues, et une plus grande valeur de *k* peut ainsi être utilisée lors de la passe suivante. Les embranchements dans le graphe deviennent alors moins nombreux, et de plus larges régions de *k*-mers faibles peuvent ainsi être corrigées. En atteignant itérativement de larges valeurs de *k*, les *reads* complets peuvent finalement être corrigés. Contrairement à FMLRC, LoRMA n'utilise cependant pas un graphe de de Bruijn d'ordre variable, et le graphe doit donc être reconstruit à chaque étape, en fonction de la valeur de *k* choisie.

Les *reads* ainsi corrigés sont alors *splittés*, en retirant les régions composées de *k*-mers faibles, et en ne conservant que les régions composées de *k*-mers solides. La correction de ces *reads splittés* est ensuite raffinée via une deuxième phase de correction, par alignement multiple. Pour cela, un graphe de de Bruijn est construit à partir des *k*-mers des *reads* corrigés, sans fixer de seuil de solidité. Les chemins simples (*i.e.* sans embranchements) du graphe sont ensuite énumérés, afin de définir le chemin du graphe dictant la séquence de chaque *read*. Chaque chemin est ainsi composé d'un ensemble de chemins simples, appelés *segments*. Pour chacun de ces segments, l'ensemble des *reads* le traversant est enregistré. Les *reads* similaires à un *read* donné peuvent ainsi être retrouvés à l'aide du graphe, via leurs *k*-mers communs, en suivant le chemin associé à ce *read*, et en sélectionnant les *reads* traversant des segments communs. Ce sous-ensemble de *reads* similaires est ensuite filtré, afin de ne conserver que les *reads* partageant un nombre suffisant de *k*-mers avec le *read* initial.

Une séquence consensus peut alors être déterminée, à partir du *read* initial et de ses *reads* similaires. Pour cela, la séquence consensus est tout d'abord définie comme étant le *read* initial lui-même. Ce consensus est ensuite mis à jour itérativement, par alignement des *reads* similaires au *read* initial. Chacun de ces *reads* similaires est ainsi considéré indépendamment, et aligné sur le consensus actuel. L'alignement multiple

courant est alors mis à jour, et le consensus est, à son tour, actualisé en fonction de l'alignement multiple, par vote majoritaire à chaque position. Ce processus est ainsi répété, jusqu'à ce que tous les *reads* similaires aient été considérés et alignés. L'alignement multiple final est alors étudié, et les positions du consensus supportées par au moins deux *reads* sont utilisées pour raffiner la correction du *read* initial.

Suite à l'étape de *splitting* réalisée après la passe de correction à l'aide des différents graphes de de Bruijn, LoRMA produit des *reads* corrigés *splittés*. De plus, lors de l'étape d'alignement multiple, les bases couvertes par moins de deux autres *reads* longs sont reportées en minuscules. Ainsi, les *reads* peuvent de nouveau être *trimmés* ou *splittés* après cette étape de correction.

3.3.2.2 Daccord, 2017

Bien que reposant également sur des graphes de de Bruijn, Daccord se différencie de LoRMA, et ne construit pas un graphe unique à partir de l'ensemble des k-mers des *reads*. En effet, un ensemble de graphes locaux est plutôt construit, sur de petites régions de sous-ensembles de *reads* similaires, définis après une étape d'alignement. Daccord commence donc par calculer les alignements entre ces *reads*, à l'aide de DALIGNER. Un alignement entre deux *reads* A et B est alors représenté par un *tuple d'alignement*, de la forme $(A_b, A_e, B_b, B_e, S, E)$, où :

- A_b et A_e représentent respectivement les positions de début et de fin de l'alignement sur A.
- B_b et B_e représentent respectivement les positions de début et de fin de l'alignement sur B.
- S représente l'orientation de B par rapport à A dans l'alignement (complémentaire inverse ou non).
- E représente la séquence des opérations d'édition (on appellera E le *script d'édition*) permettant de transformer $A[A_b..A_e]$ en $B[B_b..B_e]$, si $S = 0$, ou en $\overline{B}[B_b..B_e]$, si $S = 1$ (où $\overline{B}$ représente la séquence complémentaire inverse de B)

Un ensemble de tuples d'alignements entre un *read* A et d'autres *reads* définit alors une *pile d'alignements* pour le *read* A. La pile d'alignements d'un *read* A permet ainsi de représenter l'ensemble des *reads* s'alignant avec A, et étant nécessaires à la construction des graphes de de Bruijn locaux permettant de corriger ce *read*. Ces graphes sont définis sur de petites fenêtres de moins de 100 paires de bases de la pile d'alignements. Grâce aux informations contenues dans cette pile d'alignements, les données des *reads* devant être incluses dans les différentes fenêtres peuvent en effet facilement être extraites. De plus, seules les données des *reads* traversant l'intégralité d'une fenêtre donnée sont ici utilisées pour construire le graphe de de Bruijn y étant associé. Les *reads* dont les alignements débutent ou s'arrêtent au sein de cette fenêtre ne sont ainsi pas considérés. Par ailleurs, les fenêtres peuvent se chevaucher, et ne partitionnent donc pas nécessairement la pile d'alignements en un ensemble d'intervalles non chevauchants.

Chaque fenêtre de la pile d'alignements est alors traitée indépendamment. Pour commencer, le graphe de de Bruijn de la fenêtre est construit à partir des k-mers des facteurs des *reads* inclus dans celle-ci, et respectant les conditions décrites précédemment. Comme pour les méthodes reposant sur des graphes de de Bruijn présentées précédemment, un sommet source et un sommet cible sont définis pour guider la recherche de chemin dans le graphe, permettant de calculer la séquence consensus de la fenêtre. Le sommet source est défini, dans la partie gauche de la fenêtre, comme

le k-mer le plus fréquent en début des facteurs des *reads* inclus dans la fenêtre. De manière symétrique, le sommet cible est défini, dans la partie droite de la fenêtre, comme le k-mer le plus fréquent en fin des facteurs des *reads* inclus dans la fenêtre. Plusieurs paires de k-mers sont cependant considérées, afin de réduire le nombre de cas où aucun chemin ne pourrait être trouvé.

Contrairement aux méthodes se basant sur la recherche de chemins dans un graphe de de Bruijn présentées précédemment, le graphe n'est cependant pas parcouru de la source vers la cible ou de la cible vers la source. En effet, Daccord traverse plutôt le graphe dans les deux directions simultanément, et recherche alors des sommets communs permettant de rejoindre les deux chemins. Ces traversées peuvent ainsi produire un ensemble de chemins. Dans ce cas, le consensus de la fenêtre est choisi en alignant les séquences des chemins obtenus sur le facteur du *read* A inclus dans la fenêtre, et en choisissant la séquence ayant la plus faible distance d'édition avec ce facteur. La procédure est alors répétée avec toutes les fenêtres de la pile d'alignements du *read* à corriger.

Une fois les consensus de toutes les fenêtres calculés, le *read* peut alors effectivement être corrigé. Pour cela, le consensus associé à chaque fenêtre de la pile d'alignements est aligné sur le facteur correspondant du *read*. Les scripts d'édition permettant de transformer les facteurs du *read* A associés à chaque fenêtre en leurs consensus respectifs sont ainsi calculés. Les fenêtres sont ensuite de nouveaux considérées indépendamment, afin de définir des paires de positions, à partir de ces scripts d'édition. Pour une fenêtre commençant en position b sur le *read* A, la paire de positions $(b + i, 0)$ est assignée à la i-ème opération d'édition n'étant pas une insertion, et la paire de positions $(b + i, -d)$ est associée à la d-ème opération d'édition étant une insertion avant la i-ème opération d'édition n'étant pas une insertion, en considérant les opérations de la droite vers la gauche. Chaque paire de positions est annotée avec la base correspondante du consensus de la fenêtre. Une fois toutes les fenêtres traitées, les paires de positions sont alors triées, et le consensus du *read* est obtenu par un vote majoritaire pour chaque paire de positions, et observant les bases qui leur sont associées.

Daccord produit ainsi des *reads* corrigés *splittés*. En effet, il est possible que la recherche de chemins dans le graphe ne produise aucun consensus pour une fenêtre donnée. Dans ce cas, cette fenêtre ne peut pas être considérée lors de l'étape de réalignement des consensus sur le *read* initial, et son script d'édition, ainsi que les paires de positions associées, ne peuvent alors pas être déterminées. Ces consensus manquants se traduisent alors par des paires de positions manquantes dans la suite de paires de positions triées. La séquence consensus ne peut ainsi pas être calculée sur ces paires de positions manquantes, et de multiples séquences consensus, correspondant à des suites de paires de positions consécutives, sans paire manquante, sont alors produites indépendamment.

3.4 Synthèse

Dans ce chapitre, nous avons décrit l'état de l'art des méthodes disponibles pour la correction de *reads* longs, qu'elles emploient une approche hybride ou une approche par auto-correction. Quatre approches ont été décrites pour la correction hybride : l'alignement de *reads* courts sur les *reads* longs, l'alignement de *reads* longs et d'assemblages de *reads* courts, l'utilisation de graphes de de Bruijn, et l'utilisation de modèles de Markov cachés. Deux approches ont été décrites pour l'auto-correction :

l'alignement multiple, et l'utilisation de graphes de de Bruijn. Les principes algorithmiques de chacune des méthodes employant ces différentes approches ont ainsi été décrits. L'ensemble des méthodes de correction disponibles est résumé Table 3.1 pour la correction hybride, et Table 3.2 pour l'auto-correction. Ces tables décrivent également les approches employées par ces différentes méthodes, les sorties qu'elles proposent, leurs années de publication, ainsi que le type de *reads* (PacBio ou ONT) sur lesquels elles ont été validées, dans leurs publications respectives.

Méthode	Approche	Production de *reads*			Casse	Année	Validé sur *reads*
		Complets	*Trimmés*	*Splittés*			
PBcR	Alignement de *reads* courts	✖	✖	✔	✖	2012	PacBio
LSC	Alignement de *reads* courts	✖	✔	✖	✖	2012	PacBio
ECTools	Alignement de contigs	✖	✖	✔	✖	2014	PacBio
LoRDEC	Graphe de de Bruijn	✔	✖	✖	✔	2014	PacBio
Proovread	Alignement de *reads* courts	✔	✖	✔	✖	2014	PacBio
Nanocorr	Alignement de *reads* courts	✔	✖	✖	✖	2015	ONT
NaS	Alignement de *reads* courts	✔	✖	✖	✖	2015	ONT
CoLoRMap	Alignement de *reads* courts	✔	✖	✖	✔	2016	PacBio
Jabba	Graphe de de Bruijn	✖	✖	✔	✖	2016	PacBio
LSCplus	Alignement de *reads* courts	✔	✔	✖	✖	2016	PacBio
HALC	Alignement de contigs	✔	✔	✔	✔	2017	PacBio
HECIL	Alignement de *reads* courts	✔	✖	✖	✖	2018	PacBio
Hercules	Modèles de Markov cachés	✔	✖	✖	✖	2018	PacBio
FMLRC	Graphe de de Bruijn	✔	✖	✖	✖	2018	PacBio
MiRCA	Alignement de contigs	✖	✔	✖	✖	2018	ONT

TABLE 3.1 – **Liste des méthodes de correction hybride de *reads* longs.** Les *reads* complets correspondent aux *reads* non *trimmés* et non *splittés*. La colonne casse indique si la méthode reporte les bases corrigées dans une casse différente des bases non corrigées.

Méthode	Approche	Production de *reads*			Casse	Année	Validé sur *reads*
		Complets	*Trimmés*	*Splittés*			
PBcR-BLASR	Alignement multiple	✖	✖	✔	✖	2013	PacBio
PBDAGCon	Alignement multiple	✔	✖	✖	✖	2013	PacBio
Sprai	Alignement multiple	✔	✖	✖	✖	2014	PacBio
PBcR-MHAP (PBDAGCon)	Alignement multiple	✔	✖	✖	✖	2015	PacBio
PBcR-MHAP (FalconSense)	Alignement multiple	✖	✖	✔	✖	2015	PacBio
FalconSense	Alignement multiple	✖	✖	✔	✖	2016	PacBio
LoRMA	Graphe de de Bruijn	✖	✖	✔	✖	2016	PacBio
Sparc	Alignement multiple	✔	✖	✖	✖	2016	PacBio + ONT
Canu	Alignement multiple	✖	✖	✔	✖	2017	PacBio + ONT
Daccord	Graphe de de Bruijn	✖	✖	✔	✖	2017	PacBio + ONT
MECAT	Alignement multiple	✖	✖	✔	✖	2017	PacBio + ONT
FLAS	Alignement multiple	✖	✔	✔	✖	2019	PacBio

TABLE 3.2 – **Liste des méthodes d'auto-correction de *reads* longs.** Les *reads* complets correspondent aux *reads* non *trimmés* et non *splittés*. La colonne casse indique si la méthode reporte les bases corrigées dans une casse différente des bases non corrigées.

Chapitre 4

Évaluation des méthodes de correction de *reads* longs

Sommaire

4.1 Introduction

4.1.1 Contexte

Comme mentionné Section 2.5.3, la première étape de nombreuses applications d'analyse de données de séquençage est la correction, afin de limiter l'impact des erreurs de séquençage sur les résultats. Cette étape de correction est d'autant plus importante lorsque des *reads* longs sont utilisés, au vu des importants taux d'erreurs de ces derniers. Dans le Chapitre 3, les différences algorithmiques entre la correction de *reads* courts et la correction de *reads* longs, dues à leurs taux et à leurs profils d'erreurs extrêmement différents, ont été décrites. Les nouveaux développements algorithmiques permettant de corriger les *reads* longs y ont ainsi été présentés, en décrivant l'état de l'art des nombreuses méthodes de correction disponibles, aussi bien pour la correction hybride que pour l'auto-correction.

La qualité de la correction ayant un impact important sur les analyses sous-jacentes, il est ainsi intéressant de savoir quelle approche et quelle méthode de correction sont les plus adaptées à une expérience particulière, en fonction de la technologie de séquençage utilisée, de la profondeur de séquençage des *reads*, de leur taux d'erreurs ou encore du génome séquencé, par exemple. Ainsi, développer des méthodes permettant d'évaluer les outils de correction, en produisant des statistiques précises et fiables décrivant en détail la qualité de la correction est un besoin important.

Ces méthodes d'évaluation doivent permettre de réaliser des *benchmarks* compréhensibles et facilement reproductibles, afin d'identifier facilement et efficacement quel correcteur est le plus adapté à un cas donné. Ces méthodes doivent également être utilisables sur des jeux de données de complexité variable, aussi bien sur des génomes bactériens que sur des génomes eucaryotes, afin de couvrir une large variété des différents scénarios pouvant être rencontrés.

De plus, la correction est souvent l'étape la plus coûteuse des pipelines d'analyse de *reads* longs. Les méthodes permettant d'évaluer les outils de correction doivent ainsi être rapides et efficaces, pour permettre de comparer aisément les différents outils de correction disponibles. En particulier, ces méthodes ne doivent donc pas être beaucoup plus coûteuses, ni en temps ni en mémoire, que les outils qu'elles évaluent.

Ce dernier point est particulièrement critique, car les outils d'évaluation se placent également dans la perspective du développement de nouvelles méthodes de correction. En effet, ces outils peuvent permettre de fournir des comparaisons précises et rapides entre les différentes méthodes de correction de l'état de l'art, et ainsi permettre aux développeurs d'évaluer et de comparer facilement leurs outils.

Pour les développeurs de nouvelles méthodes de correction, aussi bien que pour les utilisateurs cherchant à identifier le correcteur le plus adapté à leurs données, les outils d'évaluation doivent donc décrire le comportement des méthodes évaluées avec précision. En particulier, ils doivent permettre de quantifier, entre autres, le nombre de bases correctement corrigées, ainsi que les erreurs additionnelles potentiellement introduites par la correction, afin d'identifier les limitations de chaque outil.

4.1.2 Travaux précédents

Les travaux introduisant de nouvelles méthodes de correction évaluent généralement la qualité de leurs outils en se basant sur la qualité d'alignement des *reads*

corrigés sur un génome de référence. Bien que cette approche soit intéressante, les informations qu'elle produit sont toutefois incomplètes. En particulier, les *reads* corrigés de très mauvaise qualité, ne pouvant s'aligner, ou les régions complexes du génome, où l'alignement est plus difficile, sont alors susceptibles de ne pas être prises en compte.

Afin de répondre à ces problématiques, LRCstats [61], un outil d'évaluation destiné aux méthodes de correction de *reads* longs, a été développé. Cet outil s'inspire notamment de travaux précédents, portant sur l'évaluation des méthodes de correction de *reads* courts [126]. Il permet ainsi d'obtenir des métriques décrivant la qualité de la correction des *reads* longs en elle-même, et ne présente donc pas uniquement les similarités entre les *reads* longs corrigés ayant pu être alignés et le génome de référence. LRCstats a été développé en se basant sur des données simulées, permettant d'identifier avec précision le type et l'emplacement des erreurs introduites.

À partir de ces données simulées, LRCstats propose ainsi de calculer un alignement de trois séquences entre les *reads* non corrigés, les *reads* corrigés, et les régions du génome de référence d'où proviennent ces *reads*. Pour cela, les fichiers générés lors de la simulation des *reads*, et décrivant les alignements entre le génome de référence et les *reads* non corrigés sont utilisés comme base de l'alignement des trois séquences. Ces alignements sont générés directement par les simulateurs, à mesure que les erreurs sont introduites dans les *reads*, et non *a posteriori* une fois les *reads* simulés. Ils décrivent donc précisément les erreurs réellement introduites lors de la simulation.

Pour calculer l'alignement des trois séquences, les *reads* corrigés sont alors alignés sur les alignements entre le génome de référence et les *reads* non corrigés, par programmation dynamique, à l'aide d'une version modifiée de l'algorithme de Needleman-Wunsch. Une fois l'alignement des trois séquences calculé, LRCstats peut alors fournir les taux d'erreurs des reads avant et après correction, ainsi que les comptes détaillés de chaque type d'erreur, pour les *reads* non corrigés et pour les *reads* corrigés, en analysant chaque colonne de l'alignement des trois séquences.

Cependant, la simple étude des taux d'erreurs des *reads* avant et après la correction ne représente pas une évaluation satisfaisante de la qualité de celle-ci. Par exemple, une telle approche n'apporte aucune information à propos de l'insertion de nouvelles erreurs par le correcteur utilisé. Pour pallier cette limitation, des métriques additionnelles telles que le *rappel* (*i.e.* le nombre de bases correctement corrigées, parmi toutes les bases nécessitant une correction) et la *précision* (*i.e.* le nombre de bases correctement corrigées, parmi toutes les bases modifiées par le correcteur), doivent être calculées, afin de mieux identifier les avantages et les inconvénients des outils de correction évalués.

De plus, LRCstats consomme d'importantes quantités de ressources quand un grand nombre de *reads* doit être évalué, comme par exemple quand la profondeur de séquençage ou la taille du génome sont élevées. Cependant, des profondeurs de séquençage importantes peuvent permettre de mieux corriger les *reads* longs [104]. De ce fait, la correction de tels jeux de données doit également pouvoir être évaluée en un temps raisonnable. De façon similaire, la méthodologie d'alignement proposée par LRCstats montre également ses limitations, notamment en termes de consommation de mémoire, quand la longueur des *reads* à évaluer augmente. De tels *reads* commençant à apparaître dans des travaux récents [46], notamment grâce aux librairies *ultra-long reads* ONT, il est également important de pouvoir évaluer les méthodes de correction sur ces *reads*.

4.1.3 Contribution

Afin de faire face aux limitations de LRCstats, nous avons développé ELECTOR [1], un nouvel outil d'évaluation destiné aux méthodes de correction de *reads* longs. ELECTOR fournit un plus large éventail de métriques que LRCstats, telles que le rappel, la précision, et le taux de bases correctes de chaque *read* corrigé, afin d'évaluer avec précision la qualité de la correction. De telles métriques avaient déjà été proposées dans des travaux précédents, et dédiés aux *reads* courts, notamment via l'outil d'évaluation mentionné Section 4.1.2. Cependant, cette contribution sort du cadre de ce travail, les algorithmes permettant de traiter les *reads* courts étant fondamentalement différents de ceux permettant de traiter des *reads* longs.

ELECTOR apporte également des informations à propos des difficultés typiques que peuvent rencontrer les correcteurs de *reads* longs, tels que les homopolymers, contenant des erreurs systématiques dans les *reads* ONT, ainsi que les *reads* ayant été *trimmés*, *splittés*, ou étendus durant le processus de correction.

Afin de fournir ces métriques additionnelles, la stratégie d'alignement de LRCstats est remplacée par une stratégie d'alignement multiple capable de passer efficacement à l'échelle, dans ELECTOR. Cette stratégie permet ainsi, comme LRCstats, de comparer les trois différentes versions de chaque *read* : le *read* non corrigé, le *read* corrigé, et le *read* de *référence*, correspondant à la région du génome de référence d'où provient le *read* original, et représentant ainsi une version parfaite de ce *read*, dans lequel aucune erreur n'aurait été introduite. À la différence de LRCstats, cet alignement multiple n'est cependant pas calculé en utilisant comme base l'alignement entre le *read* non corrigé et le génome de référence, décrit dans les fichiers produits lors de la simulation. En effet, cet alignement est plutôt calculé à l'aide d'une méthode basée sur des graphes d'alignement d'ordre partiel, permettant de produire de réels alignements multiples.

Pour permettre à cette stratégie d'alignement multiple de passer à l'échelle, aussi bien sur les *ultra-long reads* que sur des jeux de données de taille importante, nous avons également introduit une nouvelle heuristique. Celle-ci combine une approche d'ancrage aux graphes d'alignement d'ordre partiel, afin de segmenter l'alignement multiple en plusieurs instances de plus petites tailles, alors plus rapides à calculer, et moins coûteuses en mémoire. Ainsi, en plus d'apporter des métriques additionnelles, ELECTOR permet également une évaluation plus rapide, et passant plus efficacement à l'échelle, par rapport à LRCstats.

De plus, ELECTOR évalue également la qualité d'alignement des *reads* corrigés sur le génome de référence, ainsi que la qualité de l'assemblage pouvant être généré à partir de ces *reads*, et fournit donc de nouvelles métriques et de nouvelles informations, non prises en compte par LRCstats.

4.2 Vue d'ensemble

Le pipeline d'ELECTOR est divisé en deux modules indépendants, permettant d'évaluer respectivement la qualité de la correction, et la qualité d'alignement et d'assemblage des *reads* corrigés. Ce pipeline et ces deux modules sont illustrés Figure 4.1.

1. https://github.com/kamimrcht/ELECTOR

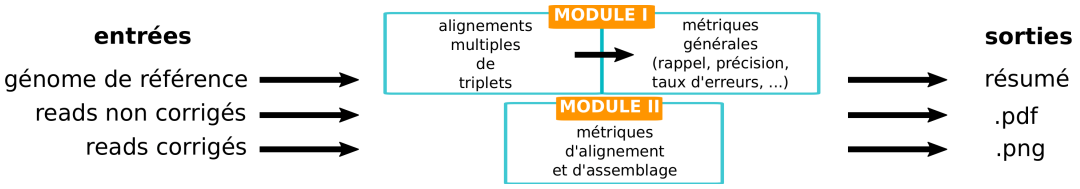

FIGURE 4.1 – **Pipeline d'ELECTOR.** Les entrées sont les *reads* à différentes étapes : sans erreurs (issus du génome de référence), avec erreurs, et corrigés (après avoir appliqué un outil de correction). Dans le premier module, un alignement multiple est calculé à partir des trois versions de chaque *read*. Les résultats sont ensuite analysés afin de fournir des métriques décrivant précisément la qualité de la correction, telles que le rappel, la précision, ou encore le taux d'erreurs des *reads* avant et après correction. Dans le second module, les *reads* corrigés sont assemblés à l'aide de Miniasm, et ces *reads*, ainsi que les contigs obtenus après l'étape d'assemblage, sont alignés sur le génome de référence, à l'aide de Minimap2. De nouvelles métriques, décrivant la qualité d'alignement des *reads* et la qualité de l'assemblage sont alors produites. Un résumé de l'ensemble de ces métriques, en format texte et en format PDF, ainsi que des graphiques illustrant les différentes métriques, sont alors produits.

ELECTOR a initialement été développé pour des *reads* simulés. Ce choix a été principalement motivé par le besoin de connaître les *reads* de *référence*, afin de contrôler précisément la qualité des résultats fournis par les méthodes de correction évaluées. Ces *reads* de référence sont des portions du génome de référence, représentant des versions parfaites des *reads* originaux, dans lesquels aucune erreur n'aurait été introduite. Cependant, des *reads* réels peuvent également être utilisés, du moment qu'un génome de référence est disponible pour l'espèce étudiée. ELECTOR prend ainsi en entrée ce génome de référence, un ensemble de *reads* corrigés, ainsi que les *reads* non corrigés correspondants, soit au format FASTA dans le cas de données réelles, soit via l'ensemble de fichiers produits par le simulateur dans le cas de données simulées. L'ensemble de ces séquences est alors passé indépendamment aux deux modules. Le pipeline d'ELECTOR reste alors globalement identique dans le cas de données simulées et dans le cas de données réelles, la seule différence étant la façon dont les *reads* de référence sont retrouvés.

Dans le cas de données simulées, ELECTOR est compatible avec les simulateurs de *reads* longs SimLoRd et NanoSim, permettant de simuler, respectivement, des *reads* PacBio et des *reads* ONT. Ainsi, quand un de ces deux outils de simulation est utilisé, les séquences de référence sont directement retrouvées par ELECTOR, en analysant les fichiers générés durant la simulation, et décrivant les erreurs introduites dans les *reads*. La compatibilité avec ces simulateurs récents et performants permet de s'assurer que des séquences simulant de manière précise les caractéristiques actuelles des *reads* longs pourront être utilisées avec ELECTOR. Cependant, ELECTOR n'est pas entièrement dépendant de ces outils, et d'autres simulateurs de *reads* longs peuvent ainsi être utilisés. Dans ce cas, les *reads* de référence ne sont cependant pas retrouvés automatiquement, et doivent donc être directement passés en entrée, avec les *reads* corrigés et les *reads* non corrigés.

Dans le cas de données réelles, les *reads* de référence sont retrouvés par ELECTOR en alignant les *reads* non corrigés sur le génome de référence, à l'aide de Minimap2. Seul le meilleur alignement de chaque *read* est conservé, et est alors utilisé pour définir le *read* de référence correspondant. Dans le cas où un *read* non corrigé ne peut pas s'aligner sur le génome de référence, et ne peut donc pas produire un *read* de référence, le *read* est alors exclus de l'évaluation. En dehors de cette étape, le pipeline d'évaluation reste inchangé, bien que, puisque les alignement peuvent clipper les *reads*, le calcul de certaines métriques varie légèrement. Ces différences

sont détaillées Section 4.3.3.1.

De plus, ELECTOR a été développé comme un outil facile d'utilisation, directement compatible avec l'intégralité des méthodes de correction présentées dans le Chapitre 3, sans nécessiter un quelconque prétraitement de la part de l'utilisateur. ELECTOR se veut également pratique pour l'utilisateur, notamment en fournissant ses résultats sous différents formats, tels que des graphiques pouvant être directement intégrés à ses projets.

4.3 Premier module

Le premier module d'ELECTOR compare les trois différentes versions de chaque *read* (référence, non corrigé, et corrigé), définissant un triplet, à l'aide d'un alignement multiple. Les différences et les similarités à chaque position de l'alignement sont alors recueillies, afin de calculer les diverses métriques d'évaluation.

4.3.1 Préparation des triplets

Les trois différentes versions d'un *read* donné sont retrouvées à l'aide de leurs *headers*. Pour cela, les fichiers contenant les différentes versions des *reads* sont triés, afin que les trois versions d'un même *read* apparaissent toujours dans le même ordre au sein de chaque fichier. Dans le cas d'un *read* ayant été *splitté* durant le processus de correction, la version non corrigée et la version de référence de ce *read* sont toutes deux dupliquées, autant de fois que le *read* corrigé comporte de fragments, afin que des triplets puissent toujours être définis.

De plus, contrairement à LRCstats, ELECTOR est directement compatible avec l'ensemble des méthodes de correction présentées dans le Chapitre 3, et listées Table 3.1 et Table 3.2. Cette compatibilité signifie que, dans le cas des méthodes de correction modifiant les *headers* des *reads* corrigés, ELECTOR est capable de retrouver automatiquement les *headers* des *reads* originaux auxquels ils correspondent, afin de définir les triplets, sans nécessiter un quelconque prétraitement de la part de l'utilisateur. Dans le cas de LRCstats, cette étape de conversion des *headers* modifiés vers les *headers* originaux doit être effectuée manuellement par l'utilisateur. Cette contrainte force donc celui-ci à connaître les règles de modification des *headers* par les différents correcteurs, et peut ainsi potentiellement limiter le nombre d'outils inclus dans un *benchmark*.

4.3.2 Alignement multiple de triplets

4.3.2.1 Principe

Pour chaque triplet, un alignement multiple est construit en utilisant un algorithme d'alignement d'ordre partiel, ajoutant séquentiellement chaque version du *read* à l'alignement multiple, en commençant avec le *read* de référence, puis y ajoutant le *read* corrigé, et enfin, le *read* non corrigé.

Cet alignement multiple est calculé à l'aide de POA [65, 64, 36], une méthode basée sur des graphes d'alignement d'ordre partiel. Ces graphes, étant des DAG, sont utilisés comme des structures de données contenant les informations des séquences alignées. POA construit ainsi l'alignement multiple itérativement, le DAG contenant, à chaque étape, le résultat de l'alignement multiple actuel. Les sommets du DAG représentent alors les nucléotides des séquences déjà alignées, et un arc est présent entre deux sommets si les nucléotides associés sont consécutifs dans une des

séquences contenues dans l'alignement multiple. Les chemins du DAG représentent ainsi les différents alignements. Chaque nouvelle séquence est alors alignée sur le DAG courant, via une généralisation de l'algorithme de Needleman-Wunsch. Les sommets et les arcs correspondant à cette nouvelle séquence sont alors fusionnés avec ceux déjà existants, afin de simplifier le graphe lorsque cela est possible. Dans le cas contraire, de nouveaux sommets et de nouveaux arcs sont ajoutés au graphe. La construction de l'alignement multiple à l'aide d'un graphe d'alignement d'ordre partiel est illustrée Figure 4.2. Une fois les trois séquences alignées, et le graphe construit, l'alignement multiple peut alors être extrait sous forme de matrice. Pour cela, le graphe est simplement parcouru, en associant chacun de ses sommets à une colonne de la matrice. Un sommet non joignable par un arc issu d'une séquence donnée se traduit alors par la présence d'un caractère de délétion («-») dans la ligne de la matrice correspondant à cette séquence. L'extraction de l'alignement multiple sous forme de matrice, à partir d'un graphe d'alignement d'ordre partiel, est illustrée Figure 4.3.

4.3.2.2 Stratégie de segmentation

La procédure d'alignement multiple présentée ci-dessus peut cependant être coûteuse en temps lorsqu'elle est appliquée à des séquences longues et bruitées telles que les *reads* longs. En effet, ces derniers contiennent de nombreuses erreurs, et impliquent ainsi de nombreux embranchements dans le graphe, et donc des étapes d'alignement coûteuses lors de l'ajout de nouvelles séquences au sein de ce graphe. De ce fait, une stratégie de segmentation a été développée pour ELECTOR, afin de permettre un calcul rapide et efficace des alignements multiples, et ainsi garantir un meilleur passage à l'échelle.

Cette stratégie consiste à diviser les alignements multiples de triplets en plusieurs alignements multiples de plus petites tailles, alors moins coûteux à calculer. En s'inspirant des approches de MUMmer et de Minimap, basées sur le problème de la plus longue sous-séquence strictement croissante, l'alignement multiple global d'un triplet est ainsi divisé en plusieurs instances, définies sur de petites fenêtres, et séparées par des régions de *matches* exacts. Cette stratégie de segmentation est illustrée Figure 4.4, et détaillée ci-dessous.

Pour chaque triplet, un ensemble de graines est calculé, à partir des k-mers :

1. Apparaissant dans chacune des trois versions du *read*.
2. N'étant répétés dans aucune des trois versions du *read*.
3. Ne se chevauchant dans aucune des trois versions du *read*.

Ces graines sont alors utilisées afin de segmenter l'alignement multiple. Pour cela, la plus longue sous-séquence de graines communes aux trois versions du *read*, notée S, est calculée à l'aide d'un algorithme de programmation dynamique. Les paires de graines successives de S définissent alors des fenêtres de régions similaires dans les trois différentes versions du *read*, séparées par des régions de *matches* exacts, étant les graines elles-mêmes. L'alignement multiple peut ainsi être segmenté, selon les fenêtres définies par ces graines. Par conséquent, les facteurs des différentes versions du *read* correspondant à ces fenêtres peuvent alors être alignés indépendamment, comme décrit dans la Section 4.3.2.1. Les graphes utilisés pour calculer les alignements multiples étant sensiblement plus petits, et contenant donc moins

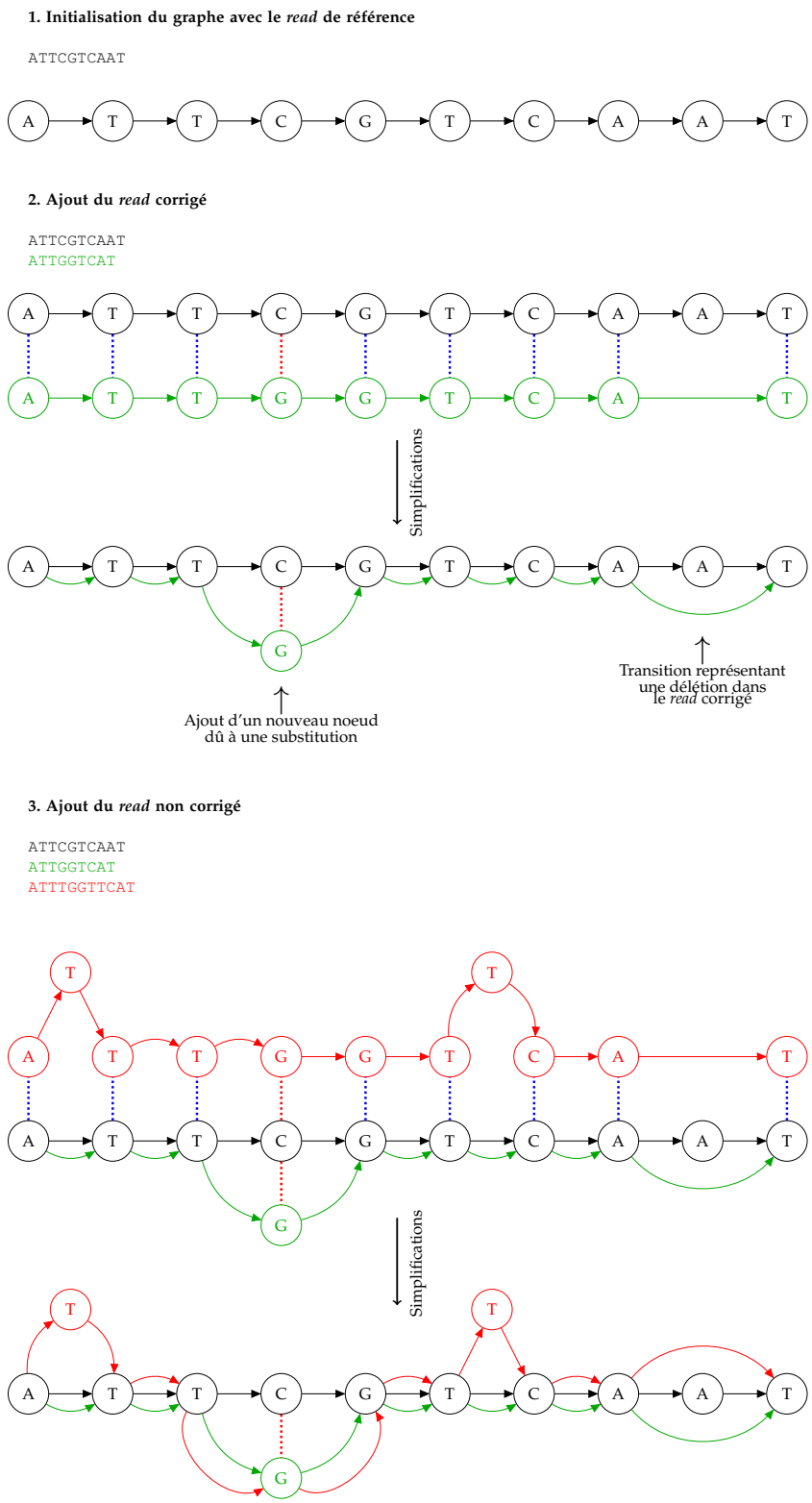

FIGURE 4.2 – **Construction de l'alignement multiple d'un triplet avec un graphe d'alignement d'ordre partiel.**

d'embranchements, les alignements multiples de ces fenêtres peuvent alors être calculés plus rapidement. Un ensemble de petits alignements multiples, correspondant aux différentes fenêtres, est ainsi obtenu. Ces derniers sont alors concaténés, avec les

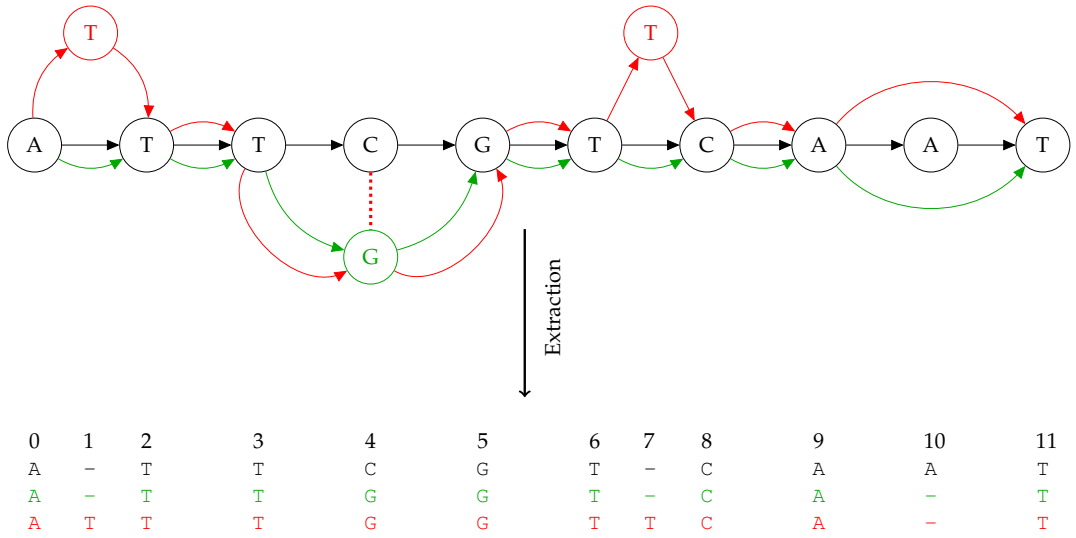

FIGURE 4.3 – **Extraction d'un alignement multiple sous forme de matrice à partir d'un graphe d'alignement d'ordre partiel.**

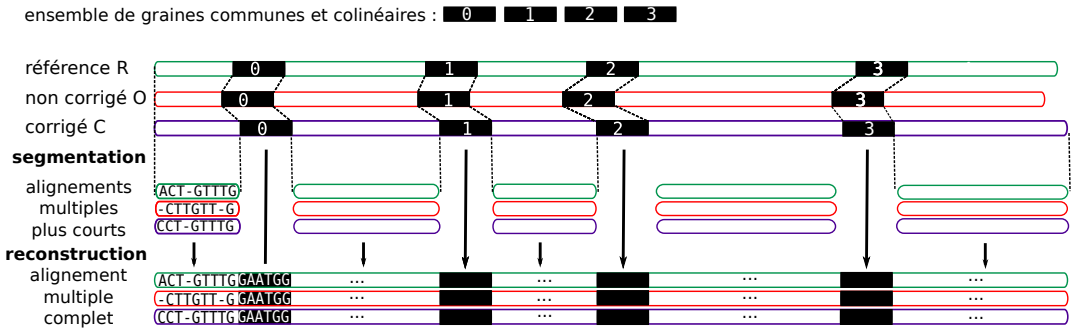

FIGURE 4.4 – **Stratégie de segmentation pour l'alignement multiple d'un triplet représentant la version de référence, la version non corrigée, et la version corrigée d'un même *read*.** Plutôt que de calculer un alignement multiple sur toute la longueur des trois séquences, le problème est divisé en de plus petits alignements multiples indépendants. Puisque chaque version du *read* est différente, des *k*-mers communs (en noir) aux trois séquences, et définissant des *matches* exacts locaux, sont utilisés comme graines afin de déterminer la façon dont l'alignement multiple peut être divisé. Des alignements multiples locaux et indépendants sont alors calculés pour les facteurs des séquences bordés par des graines, ainsi que pour leurs préfixes et pour leurs suffixes. Les alignements multiples des fenêtres ainsi définies, et les *k*-mers ayant servi de graines, sont ensuite concaténés, afin d'obtenir un alignement multiple unique, sur toute la longueur des séquences.

k-mers ayant servis de graines, afin de reconstruire un alignement multiple unique, sur toute la longueur du triplet initial.

La taille des *k*-mers utilisés comme graines est adaptée en fonction des taux d'erreurs observés dans les *reads*, et est ainsi comprise entre 9 et 15. De plus, puisqu'il est difficile de choisir une valeur de *k a priori*, une stratégie itérative a été développée, afin de pouvoir tester plusieurs valeurs de *k* différentes, et choisir la valeur la plus adaptée à un triplet donné. Pour cela, la plus longue sous-séquence S de graines communes aux trois séquences est calculée pour toutes les valeurs de *k* comprises entre 9 et 15, afin de sélectionner celle minimisant la distance maximale entre deux graines. En effet, minimiser cette distance est important, car une longue distance entre deux graines implique de devoir calculer l'alignement multiple de larges facteurs des trois séquences, et donc un temps d'exécution plus important. Minimiser la distance maximale entre deux graines permet ainsi d'éviter de devoir calculer de tels alignements multiples, en s'assurant que les fenêtres définies par S soient de tailles raisonnables.

De plus, pour éviter de calculer des métriques sur des *reads* corrigés de très mauvaise qualité, certains triplets sont filtrés avant même l'étape d'alignement multiple, et les métriques leur étant associées ne sont donc pas calculées. Ainsi, les *reads* corrigés dont la longueur est plus courte que ℓ% de la longueur de leurs *reads* de référence sont filtrés, ℓ étant un paramètre initialisé à 10 par défaut. De la même façon, les triplets pour lesquels un nombre insuffisant de graines a été trouvé sont également filtrés, afin d'éviter de calculer des alignements multiples sur des séquences trop divergentes. Ces deux types de *reads* filtrés sont reportés par ELECTOR, respectivement en tant que *reads* corrigés trop courts, et que *reads* corrigés de mauvaise qualité. L'impact du choix du paramètre ℓ, dont la valeur est laissée à l'utilisateur, est discuté Section 4.5.2.

4.3.2.3 Prise en compte des *reads* de longueurs différentes dans la stratégie de segmentation

Comme décrit dans le Chapitre 3, la plupart des méthodes de correction ont tendance à *trimmer* ou à *splitter* les *reads* corrigés. Ces différents cas doivent donc être détectés afin de permettre à la stratégie de segmentation de conserver son efficacité.

Dans le cas d'un *read trimmé*, le *read* corrigé est plus court que le *read* de référence et que le *read* non corrigé, et dispose donc d'un préfixe ou d'un suffixe manquant par rapport à ces derniers. Dans le cas d'un préfixe manquant, la première fenêtre sélectionnée par la stratégie de segmentation contient alors de longs préfixes du *read* non corrigé et du *read* de référence, et un court préfixe du *read* corrigé. De telles fenêtres apparaissent à cause de la façon dont les graines sont définies, à partir des *k*-mers communs aux trois versions du *read*.

Calculer un alignement multiple entre les séquences de cette fenêtre ne serait alors pas pertinent, la séquence provenant du *read* corrigé étant sensiblement plus courte, et donc non comparable aux deux autres séquences. De plus, calculer un alignement entre les deux longues séquences du *read* non corrigé et du *read* de référence serait coûteux.

Afin de régler ce problème, ces cas sont détectés en analysant la longueur des séquences avant le calcul de l'alignement multiple. Ainsi, si au sein de la première fenêtre définie par la stratégie de segmentation, les séquences provenant du *read* non corrigé et du *read* de référence sont bien plus longues que la séquence provenant du *read* corrigé, une seconde stratégie de segmentation est mise en place. Celle-ci utilise alors uniquement les *k*-mers des séquences du *read* de référence et du *read*

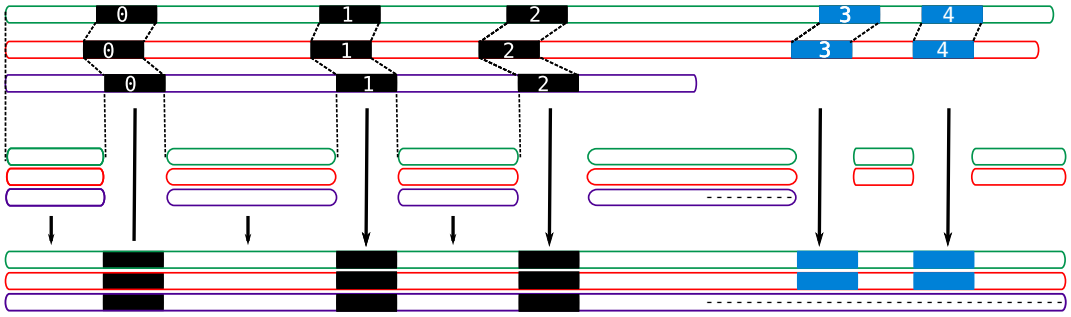

FIGURE 4.5 – **Stratégie de segmentation pour l'alignement multiple d'un triplet dont la version corrigée du *read* est plus courte.** Le *read* corrigé a ici un suffixe manquant par rapport aux deux autres *reads*. Afin d'éviter de calculer un alignement complet sur les longs suffixes du *read* de référence et du *read* non corrigé, suivant la graine numéro 2, une seconde stratégie de segmentation est appliquée, cette fois uniquement sur ces deux séquences. Celle-ci permet de calculer un nouvel ensemble de graines (graines 3 et 4, en bleu). Ces graines permettent alors de segmenter l'alignement de ces deux séquences, en définissant des fenêtres sur lesquelles des alignements locaux sont calculés indépendamment. Comme précédemment, l'alignement complet est ensuite reconstruit par concaténation de ces alignements locaux, et des k-mers ayant servi de graines. Des caractères «-» sont alors ajoutés pour compléter la fin de la ligne de l'alignement multiple correspondant au *read* corrigé, afin de représenter les délétions dans cette séquence.

non corrigé. Comme précédemment, des alignements locaux sont alors calculés sur les fenêtres définies par les nouvelles graines obtenues, et l'alignement global est reconstruit par concaténation des alignements de ces fenêtres, et des graines ayant été utilisées pour la segmentation. La ligne de l'alignement multiple correspondant au *read* corrigé est alors complétée avec des caractères «-», sur toute la longueur de son préfixe manquant, afin de représenter les délétions dans cette séquence.

De cette façon, l'alignement multiple peut être calculé efficacement, même quand les *reads* corrigés ne couvrent pas toute la longueur de leurs séquences originales. Cette procédure de segmentation additionnelle est donc extrêmement importante, en particulier pour les correcteurs produisant de nombreux *reads trimmés* ou *splittés*. En effet, sans cette stratégie, de tels *reads* induiraient d'importants temps d'exécution, dus aux calculs d'alignements de longues séquences des *reads* non corrigés et des *reads* de référence.

La procédure, ici présentée dans le cas d'un préfixe manquant, est symétrique dans le cas d'un suffixe manquant dans le *read* corrigé. Ce cas est illustré Figure 4.5. Dans le cas du traitement d'un fragment provenant d'un *read* splitté, la procédure est alors identique, et appliquée aussi bien pour le préfixe manquant que pour le suffixe manquant de ce fragment.

4.3.3 Calcul des métriques d'évaluation à partir de l'alignement multiple

4.3.3.1 Classification des *reads* corrigés

Comme indiqué précédemment, différentes catégories de *reads* corrigés sont reportées par ELECTOR. Ces catégories sont déduites à partir de l'alignement multiple, tel que décrit ci-dessous et illustré Figure 4.6.

Reads trimmés ou *splittés*

Ces *reads* sont composés d'un ou plusieurs fragments provenant d'un unique *read* original, n'ayant pu être corrigé que sur une ou plusieurs régions, alors reportées séparément. Comme indiqué Section 4.3.2.3, ces *reads* disposent d'un préfixe et / ou d'un suffixe manquant par rapport aux *reads* de référence et aux *reads* non corrigés. Pour ces *reads*, les métriques d'évaluation ne sont pas calculées sur ces préfixes ou suffixes manquants. Tous les fragments provenant d'un même *read* initial sont cependant regroupés, et le nombre de *reads trimmés* ou *splittés* est alors reporté. Pour chaque *read trimmé* ou *splitté*, la longueur totale non corrigée du *read* de référence (*i.e.* la longueur couverte par aucun fragment) est également calculée et reportée. Ces deux cas sont illustrés dans les deux premiers scénarios de la Figure 4.6.

Reads étendus

Ces *reads* disposent d'un préfixe et / ou d'un suffixe additionnel n'étant pas présent dans les *reads* de référence et dans les *reads* non corrigés. De tels *reads* peuvent, par exemple, provenir de méthodes de correction basées sur des graphes, ayant continué la traversée du graphe après avoir atteint les extrémités des *reads* non corrigés. Pour ces *reads*, les métriques d'évaluation ne sont pas calculées sur les régions étendues. Le nombre de *reads* étendus est cependant reporté, ainsi que la longueur moyenne des extensions, par rapport aux *reads* de référence. Ce cas est illustré dans le troisième scénario de la Figure 4.6.

Reads clippés

Ces *reads* proviennent d'un jeu de données réelles, et disposent d'extrémités ayant été clippées durant l'étape d'alignement des *reads* non corrigés sur le génome de référence. Cette catégorie de *reads* ne peut donc survenir que dans le cas de données réelles, les *reads* de référence étant déterminés par alignement uniquement dans ce cas. Pour ces *reads*, les métriques d'évaluation ne sont pas calculées sur les régions clippées, ces dernières n'ayant pas pu être correctement alignées sur le génome de référence, et n'ayant donc pas été utilisées pour déterminer les *reads* de référence. Les *reads* non corrigés et les *reads* corrigés disposent alors d'un préfixe et / ou d'un suffixe additionnel, n'étant pas présent dans les *reads* de référence. Ce cas est illustré dans le quatrième scénario de la Figure 4.6.

Reads de mauvaise qualité

Cette catégorie regroupe les *reads* pour lesquels un nombre insuffisant de graines a été déterminé lors de l'étape de segmentation. Comme mentionné Section 4.3.2.2, ces *reads* sont ainsi filtrés avant l'étape d'alignement multiple, et les métriques les concernant ne peuvent alors pas être calculées. Le nombre de ces *reads* est cependant reporté.

Reads trop courts

Cette catégorie regroupe les *reads* dont la longueur est plus courte que ℓ% (ℓ étant un paramètre initialisé à 10) de la longueur de leurs *reads* de référence. De nouveau, comme mentionné Section 4.3.2.2, ces *reads* sont filtrés avant l'étape d'alignement

multiple, et les métriques les concernant ne sont alors pas calculées. Comme pour les *reads* de mauvaise qualité, le nombre de ces *reads* est cependant reporté.

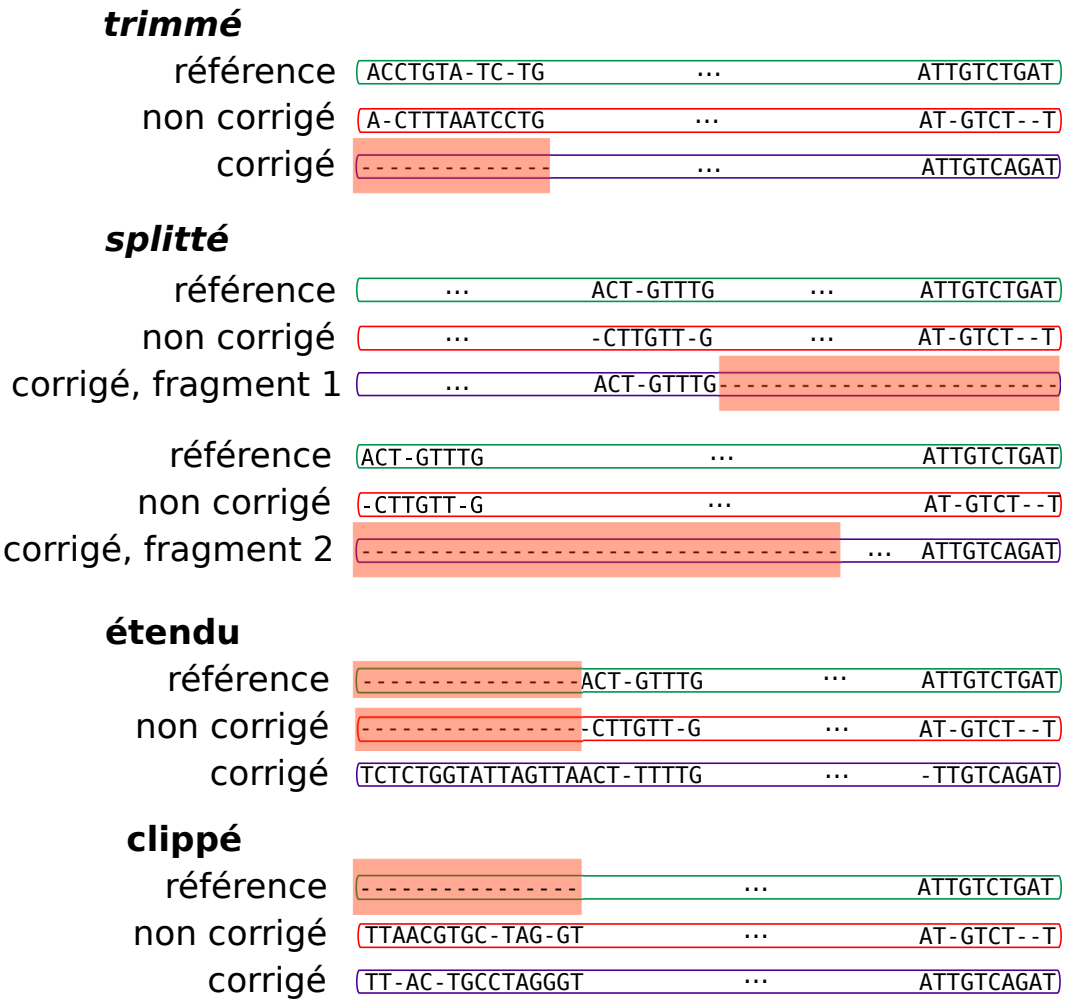

FIGURE 4.6 – **Quatre catégories de *reads* corrigés dans un alignement multiple.** Les *reads* corrigés *trimmés* et / ou *splittés* ont un préfixe et / ou un suffixe manquant par rapport à leurs *reads* non corrigés et à leurs *reads* de référence. Les *reads* corrigés étendus ont un préfixe et / ou un suffixe additionnel par rapport à leurs *reads* non corrigés et à leurs *reads* de référence. Les *reads* de référence clippés ont un préfixe et / ou un suffixe manquant par rapport à leurs *reads* non corrigés et à leurs *reads* corrigés.

4.3.3.2 Rappel, précision, taux d'erreurs

Une fois l'alignement multiple d'un triplet calculé, ce dernier fournit alors une comparaison base à base des trois versions du *read* correspondant, soulignant ainsi leurs différences et leurs similarités. Cet alignement multiple peut alors être utilisé afin de calculer les métriques d'évaluation. Le calcul du rappel, de la précision, et du taux d'erreurs est détaillé ci-dessous. Le calcul du rappel et de la précision est également illustré Figure 4.7.

Soient $nt(R, p_i)$, $nt(C, p_i)$ et $nt(O, p_i)$ les caractères respectifs du *read* de référence, du *read* corrigé, et du *read* non corrigé (ou *read original*), à la position p_i, $0 \leq i < N$ d'un alignement multiple de taille N.

Soient :

- $\mathcal{P}$ l'ensemble des positions à corriger, contenant les positions p_i telles que $nt(R, p_i) \neq nt(O, p_i)$.
- $\mathcal{E}$ l'ensemble des positions à considérer dans le *read* corrigé, contenant les positions p_i du *read* corrigé n'étant pas situées sur une région *trimmée*, *splittée*, étendue ou clippée.
- $\mathcal{C}$ l'ensemble des positions traitées, défini comme $\mathcal{P} \cup \{p_j / nt(C, p_j) \neq nt(R, p_j)\} \cap \mathcal{E}$.
- $\mathcal{C}o$ l'ensemble des positions correctes, défini comme $\mathcal{C} \cap \{p_j / nt(C, p_j) = nt(R, p_j)\}$.

Le rappel, la précision, et le taux d'erreurs sont alors calculés comme suit, avec c représentant la longueur du *read* corrigé.

$$Rappel = \frac{card(\mathcal{C} \cap \mathcal{P})}{card(\mathcal{P})} \tag{4.1}$$

$$Prcision = \frac{card(\mathcal{C}o \cap \mathcal{C})}{card(\mathcal{C})} \tag{4.2}$$

$$Taux\,d'erreurs = 1 - \frac{card(\mathcal{C}o)}{\sum_{i=0}^{c-1} i} \tag{4.3}$$

Ainsi, le rappel, la précision, et le taux d'erreurs ne sont calculés que sur les régions des *reads* ayant pu être corrigées. En particulier, ces métriques ne sont pas calculées sur les régions des *reads* originaux ou des *reads* de référence n'existant pas dans les *reads* corrigés, et pouvant apparaître lors du traitement de *reads trimmés* ou *splittés*. Dans le cas particulier des *reads splittés*, produisant un alignement multiple de triplet par fragment, une procédure particulière a été mise en place.

Les régions non corrigées sont identifiées en repérant les longues suites de caractères «-», représentant des délétions, dans les lignes des alignements multiples correspondant aux *reads* corrigés. Seules les longues suites de caractères «-», de longueur supérieure à un seuil donné, sont considérées comme des régions non corrigées, afin de conserver les plus petites régions, représentant des délétions par rapport aux deux autres *reads*. Les autres régions sont alors considérées comme corrigées. Les régions corrigées provenant des différents triplets d'un même *read* original sont alors associées en un alignement multiple unique, et les larges régions composées de délétions dans la ligne correspondant au *read* corrigé sont de nouveau détectées. Le rappel, la précision, et le taux d'erreurs sont alors calculés sur les régions

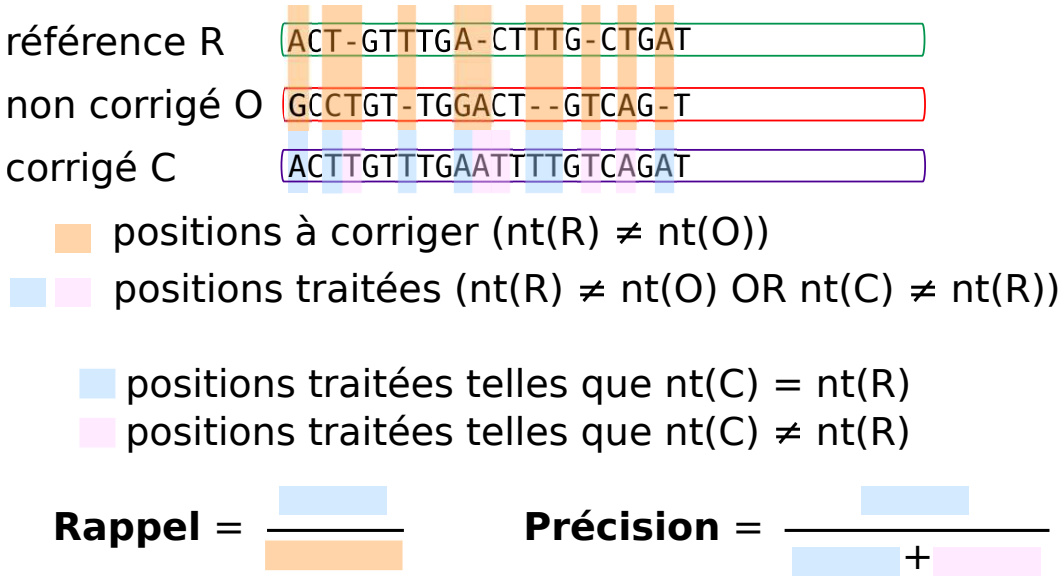

FIGURE 4.7 – **Calcul du rappel et de la précision à partir de l'alignement multiple d'un triplet.** À chaque position de l'alignement multiple, une comparaison base à base des trois versions du *read* est effectuée. $nt(R)$, $nt(C)$ et $nt(O)$ représentent respectivement les caractères du *read* de référence, du *read* corrigé et du *read* non corrigé (original), au sein l'alignement multiple, à une position donnée.

considérées comme corrigées au sein de l'alignement multiple final. La longueur manquante du *read* corrigé est, quant à elle, calculée en dénombrant les bases du *read* de référence incluses dans des régions considérées comme non corrigées. Cette procédure est illustrée Figure 4.8.

4.3.3.3 Métriques additionnelles

Comme indiqué Section 4.3.3.1, le premier module d'ELECTOR fournit également des informations sur le nombre de *reads trimmés* ou *splittés*, et sur la longueur manquante moyenne de ces *reads*, comparés à leurs *reads* de référence, ainsi que sur le nombre de *reads* étendus, et sur la longueur d'extension moyenne de ces *reads*, comparés à leurs *reads* de référence. Les *reads* corrigés trop courts par rapport à leurs *reads* de référence sont également reportés, ainsi que les *reads* corrigés considérés comme étant de trop mauvaise qualité, et n'ayant pas permis de mettre en évidence un nombre suffisant de graines lors de l'application de la stratégie de segmentation. La distribution de la longueur des *reads*, avant et après la correction, est également fournie sous la forme d'un graphique. Dans le cas de *reads* splittés, la longueur de chaque fragment est reportée indépendamment dans la distribution. Un exemple d'un tel graphique est illustré Figure 4.9. De la même façon, la distribution du rappel, de la précision, et du taux de bases correctes de chaque *read* est également fournie sous la forme d'un graphique. Un exemple est illustré Figure 4.10.

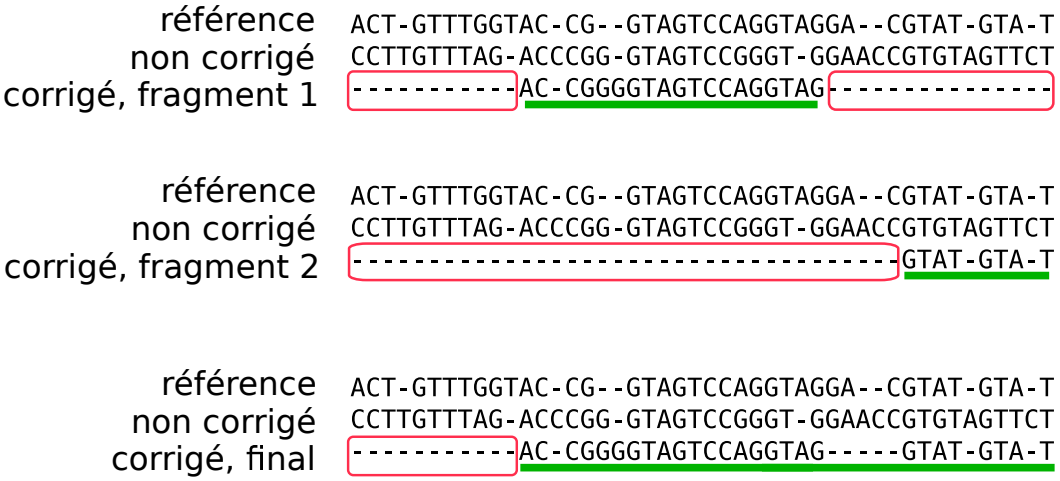

FIGURE 4.8 – **Calcul du rappel, de la précision, et du taux d'erreurs dans le cas d'un *read* corrigé *splitté*.** Ici, le *read* a été corrigé en deux fragments distincts, donnant ainsi lieu à deux alignements multiples de triplets distincts. Les larges régions non corrigées, composées uniquement de délétions dans les lignes correspondant aux *reads* corrigés, et plus longues qu'un seuil donné, sont détectées. Ces régions sont ici représentées dans des cadres rouges. Les régions restantes des alignements sont alors considérées comme corrigées. Ces régions sont ici soulignées en vert. Les régions corrigées des différents alignements multiples sont alors combinées en un alignement multiple unique (partie basse de la figure). Les larges régions non corrigées sont ensuite à nouveau détectées au sein de ce nouvel alignement multiple. Le rappel, la précision, et le taux d'erreurs sont alors calculés sur les régions corrigées finales ainsi obtenues, ici, l'unique région soulignée en vert. La longueur manquante du *read* corrigé est, quant à elle, calculée à partir du nombre de bases du *read* de référence incluses dans les régions non corrigées finales, ici, l'unique région représentée dans le cadre rouge.

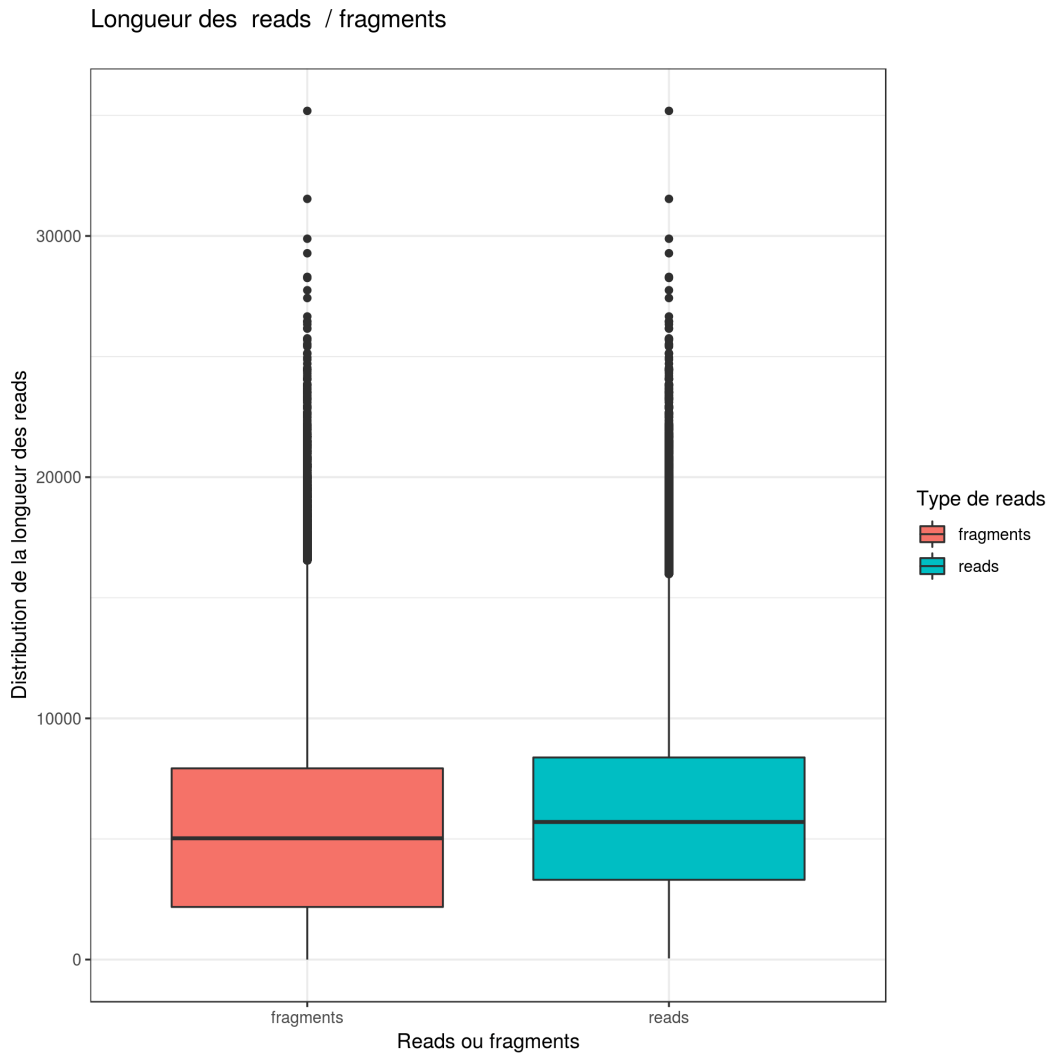

FIGURE 4.9 – **Graphique représentant la distribution de la longueur des *reads*, avant et après correction, tel que produit par ELECTOR.** Dans le cas de *reads* splittés, les *reads* sont composés de plusieurs fragments. Afin d'être plus informatif ELECTOR reporte donc la distribution de la longueur des fragments et la distribution de la longueur des *reads*.

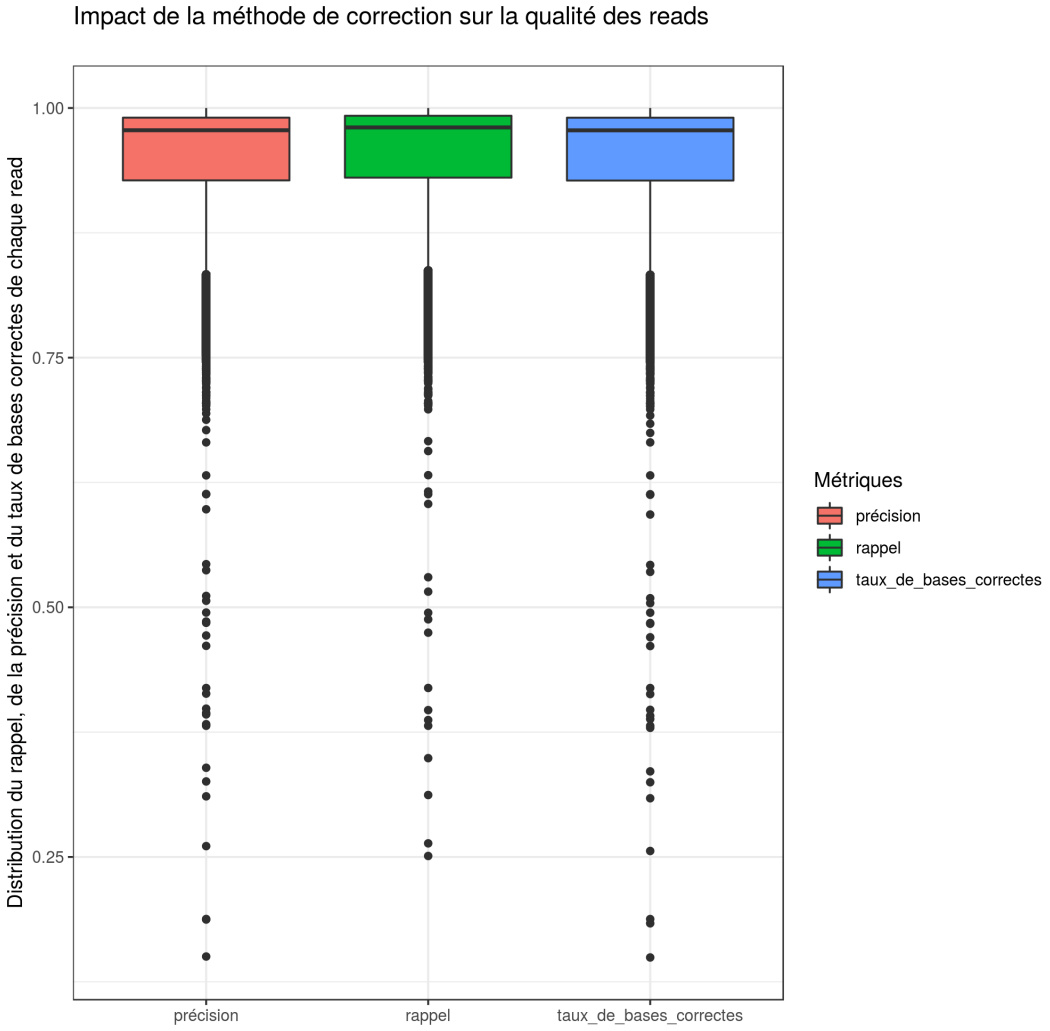

FIGURE 4.10 – **Graphique représentant la distribution du rappel, de la précision, et du taux de bases correctes, tel que produit par ELECTOR.**

Les comptes précis des erreurs d'insertions, de délétions, et de substitutions, avant et après la correction, sont également reportés. Comme pour le rappel, la précision, et le taux d'erreurs, ces comptes d'erreurs ne sont pas calculés sur les parties de l'alignement multiple correspondant à des régions *trimmées*, *splittées*, étendues, ou clippées des *reads*. Le calcul du compte de chaque type d'erreur est illustré Figure 4.11. Le taux de GC est également reporté, dans les *reads* de référence et dans les *reads* corrigés.

Enfin, le ratio de la longueur des homopolymers dans les *reads* corrigés par rapport à la longueur des homopolymers dans les *reads* de référence est également calculé. Cette métrique est particulièrement intéressante lors de l'évaluation d'un outil de correction sur des *reads* ONT, contenant des erreurs systématiques dans les homopolymers. Ainsi, plus le ratio calculé est proche de 1, plus l'outil de correction est parvenu à corriger efficacement ces erreurs. Le calcul de ce ratio est illustré Figure 4.12.

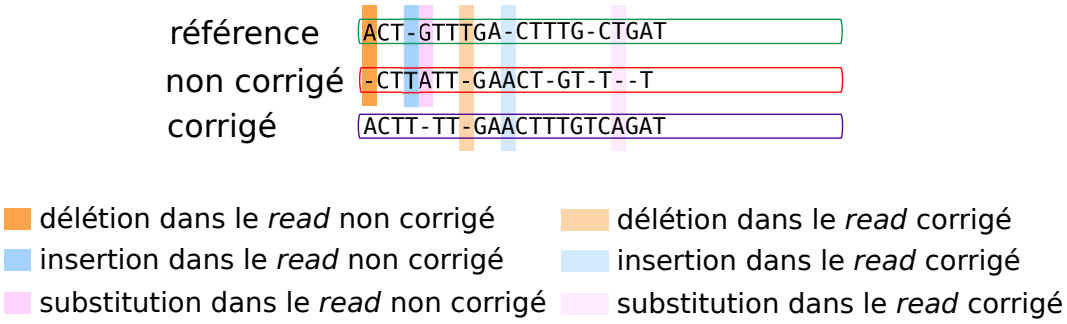

FIGURE 4.11 – **Calcul des comptes d'erreurs à partir de l'alignement multiple d'un triplet.** Pour obtenir les comptes d'insertions, de délétions, et de substitutions, le *read* corrigé et le *read* de référence, puis le *read* non corrigé et le *read* de référence sont comparés base à base, indépendamment.

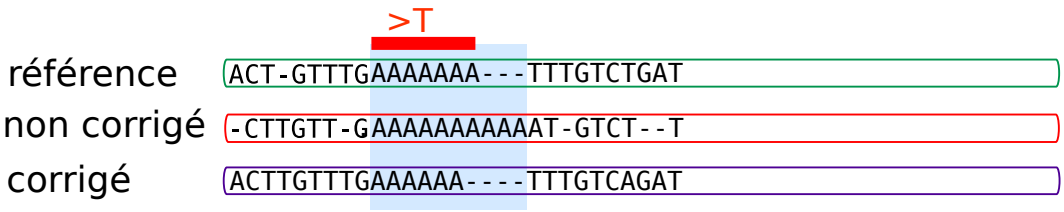

FIGURE 4.12 – **Calcul du ratio de la longueur des homopolymers à partir de l'alignement multiple d'un triplet.** Les homopolymers sont détectés comme des facteurs composés d'un unique nucléotide répété plus qu'un certain nombre de fois (6 en pratique) dans le *read* de référence. Un ratio de la longueur de ces homopolymers dans les *reads* corrigés, par rapport à leur longueur dans les *reads* de référence est alors calculé et reporté par ELECTOR. Dans cet exemple, ce ratio est de 6/7.

4.4 Second module

Le second module d'ELECTOR évalue la qualité d'alignement des *reads* corrigés sur le génome de référence, ainsi que la qualité de l'assemblage pouvant être généré à partir des *reads* corrigés. Ces deux processus additionnels d'évaluation sont présentés ci-dessous.

4.4.1 Alignement des *reads* corrigés

L'alignement des *reads* corrigés sur le génome de référence est réalisé à l'aide de Minimap2. À partir du fichier SAM généré, le nombre de *reads* corrigés produits, le nombre total de bases de ces *reads*, leur longueur moyenne, la proportion de *reads* alignés, l'identité moyenne des alignements, ainsi que la couverture du génome (*i.e.* la proportion de bases du génome de référence sur lesquelles au moins un nucléotide est aligné) sont calculés et reportés.

4.4.2 Assemblage des *reads* corrigés

Les *reads* corrigés sont assemblés à l'aide de Miniasm. D'autres assembleurs, plus précis, tels que Canu, Smartdenovo ou encore FALCON auraient pu être utilisés, mais de tels assembleurs sont plus coûteux, principalement en termes de temps d'exécution. Une comparaison des assembleurs sus-mentionnés, sur les jeux de données de la Table 4.4, après correction avec HALC, est présentée Table 4.1. FALCON

Jeu de données	Assembleur	Nombre de contigs	Temps	Mémoire (Mo)
A. baylyi	Canu	8	3 min	2 124
	Miniasm	12	17 sec	607
	Smartdenovo	6	3 min	6 695
E. coli	Canu	23	3 min	2 400
	Miniasm	23	20 sec	863
	Smartdenovo	7	6 min	6 109
S. cerevisiae	Canu	159	6 min	3 036
	Miniasm	158	1 min	2 110
	Smartdenovo	85	7 min	4 472
C. elegans	Canu	7 198	51 min	5 707
	Miniasm	2 183	7 min	11 377
	Smartdenovo	1 573	58 min	3 970

TABLE 4.1 – **Comparaison des performances de Canu, FALCON, Miniasm et Smartdenovo.** Les expériences ont été réalisées sur les jeux de données de la Table 4.4, après correction avec HALC. Tous les assembleurs ont été lancés sur 16 threads.

est cependant exclus de cette comparaison, puisque n'ayant pu être exécuté sur aucune des machines ayant servi à réaliser les différences expériences. Ces résultats montrent que, bien que la consommation de mémoire soit plus élevée que celle des autres outils sur le jeu de donnés *C. elegans*, Miniasm offre un gain considérable en termes de temps d'exécution. ELECTOR ayant été développé comme un outil permettant d'évaluer rapidement la qualité d'un outil de correction, Miniasm offre donc un bon compromis entre rapidité et précision.

Concernant les métriques d'assemblage, le nombre total de contigs produits, le nombre de contigs alignés sur le génome de référence, le nombre de *breakpoints* des contigs alignés, les tailles NGA50 et NGA75, ainsi que la couverture du génome sont reportés. Comme pour l'alignement des *reads* corrigés, les contigs sont alignés sur le génome de référence à l'aide de Minimap2, et les métriques sont calculées à partir du fichier SAM ainsi généré.

4.5 Résultats

4.5.1 Validation de la stratégie de segmentation

Afin de valider la stratégie de segmentation pour l'alignement multiple, nous étudions dans quelle mesure les résultats obtenus en appliquant cette stratégie diffèrent des résultats obtenus via un alignement multiple classique. Nous comparons donc les résultats produits par ELECTOR, en utilisant la stratégie de segmentation, et en utilisant l'implémentation classique des graphes d'alignement d'ordre partiel. Afin de valider la stratégie de segmentation, les métriques produites par ces deux approches doivent alors être extrêmement similaires, et un important gain de temps doit être obtenu via l'approche utilisant la stratégie de segmentation.

Trois expériences sont présentées, sur des jeux de données simulées à partir du génome d'*E. coli*, en faisant varier la longueur des *reads*, afin d'impacter le temps d'exécution. Les trois jeux de données de ces expériences ont été simulés avec SimLoRD, en affectant aux *reads* un taux d'erreurs de 10%, une profondeur de séquençage de 100x, et des longueurs fixées à 1 000, 10 000 et 100 000 paires de bases. Les *reads* ont ensuite été corrigés avec le module de correction de Canu, en utilisant les paramètres par défaut, avant de lancer les procédures d'évaluation.

Expérience	Rappel (%)	Précision (%)	Taux d'erreurs (%)	Temps
1k sans segmentation	99,712	98,996	98,980	2h 05 min
1k avec segmentation	99,769	98,992	98,979	28 min
10k sans segmentation	99,921	99,781	99,794	20 h 50 min
10k avec segmentation	99,921	99,795	99,793	29 min
100k sans segmentation	99,913	99,925	99,956	8 jours 18 h 38 min
100k avec segmentation	99,924	99,903	99,902	1 h 11 min

TABLE 4.2 – **Comparaison des deux approches d'alignement multiple.** La première approche utilise la stratégie de segmentation, et l'autre, l'implémentation classique des graphes d'alignements d'ordre partiel. Les différents jeux de données ont été simulés à partir du génome d'*E. coli*, avec SimLoRD, en affectant aux *reads* un taux d'erreurs de 10%, une profondeur de séquençage de 100x, et des longueurs fixées à 1 000 (1k), 10 000 (10k) et 100 000 (100k) paires de bases. Les *reads* ont été corrigés avec le module de correction de Canu, en utilisant les paramètres par défaut. Les deux approches d'alignement multiple ont été lancées sur 20 threads.

Les résultats de ces expériences sont décrits Table 4.2, et montrent que les métriques calculées via l'approche utilisant la stratégie de segmentation ne diffèrent que très légèrement des métriques calculées via l'approche d'alignement multiple classique. En revanche, en utilisant la stratégie de segmentation, un gain de temps considérable est atteint. En particulier, sur le jeu de données composé de *reads* de 100 000 paires de bases, la stratégie de segmentation permet de diviser par 171 le temps d'exécution, passant alors de plus de huit jours à légèrement plus d'une heure.

4.5.2 Impact des paramètres

Nous étudions ici l'impact du choix de la valeur du paramètre ℓ, déterminant la longueur minimale des *reads* à considérer, par rapport à leurs séquences de référence, sur les résultats produits par ELECTOR. La valeur de k utilisée lors de l'étape de segmentation étant choisie dynamiquement par ELECTOR, indépendamment pour chaque triplet, ℓ est le seul paramètre laissé au choix de l'utilisateur. Des expériences faisant varier la valeur de ℓ entre 1, 5, 10 et 15 sont présentées Table 4.3. Ces expériences ont été réalisées sur le jeu de données *E. coli* de la Table 4.4, après correction avec HALC. Ces expériences démontrent qu'en dehors des métriques décrivant le nombre de bases et la taille manquante moyenne au sein des *reads trimmés* ou *splittés*, directement impactées par le nombre de *reads* traités, la valeur de ℓ n'impacte que peu les résultats. En particulier, pour $\ell = 1$ ou $\ell = 15$, les valeurs reportées pour les taux d'erreurs, le rappel, et la précision, ne varient, au plus, que de quelques centièmes. En revanche, choisir une valeur de ℓ plus élevée permet de réduire le temps d'exécution d'ELECTOR. Bien que n'ayant qu'un impact modéré sur les expériences présentées ici, le choix d'une valeur de ℓ relativement élevée peut ainsi permettre de grandement réduire les temps d'exécution d'expériences de plus grande envergure, sur de plus grands ensembles de *reads* ou sur des génomes plus complexes.

4.5.3 Comparaison avec LRCstats

4.5.3.1 Comparaison des métriques

Nous comparons les métriques produites par ELECTOR et par LRCstats sur plusieurs jeux de données simulées de diverses espèces de taille et de complexité croissantes (*A. baylyi*, *E. coli*, *S. cerevisiae* et *C. elegans*). Les *reads* longs composant ces jeux

Valeur de ℓ	Métrique	*Reads* non corrigés	*Reads* corrigés
$\ell = 1$	Nombre de bases	92 749 586	81 320 556
	Taux d'erreurs	14,1611	0,1593
	Délétions	2 040 884	119 791
	Insertions	11 074 408	15 257
	Substitutions	1 532 186	14 116
	Rappel	-	99,9917
	Précision	-	99,8442
	Reads trimmés ou *splittés*	-	3 816
	Taille manquante moyenne	-	568,8
	Reads étendus	-	24
	Taille moyenne d'extension	-	31,1
	Reads de mauvaise qualité	-	73
	Reads trop courts	-	2
	Ratio taille homopolymers	-	1,0000
	Temps	30 min 8 sec	
$\ell = 5$	Nombre de bases	92 609 035	81 310 017
	Taux d'erreurs	14,1613	0,1567
	Délétions	2 040 581	117 994
	Insertions	11 072 813	15 084
	Substitutions	1 531 970	13 980
	Rappel	-	99,9917
	Précision	-	99,8467
	Reads trimmés ou *splittés*	-	3 816
	Taille manquante moyenne	-	567,1
	Reads étendus	-	24
	Taille moyenne d'extension	-	31,1
	Reads de mauvaise qualité	-	14
	Reads trop courts	-	99
	Ratio taille homopolymers	-	1,0000
	Temps	29 min 57 sec	
$\ell = 10$	Nombre de bases	91 950 978	81 199 351
	Taux d'erreurs	14,1520	0,1537
	Délétions	2 036 924	115 973
	Insertions	11 054 008	15 034
	Substitutions	1 528 672	13 764
	Rappel	-	99,9918
	Précision	-	99,8497
	Reads trimmés ou *splittés*	-	3 816
	Taille manquante moyenne	-	547,3
	Reads étendus	-	24
	Taille moyenne d'extension	-	31,1
	Reads de mauvaise qualité	-	2
	Reads trop courts	-	254
	Ratio taille homopolymers	-	1,0000
	Temps	27 min 52 sec	
$\ell = 15$	Nombre de bases	91 312 640	81 036 912
	Taux d'erreurs	14,1483	0,1514
	Délétions	2 032 455	114 146
	Insertions	11 029 894	14 996
	Substitutions	1 524 974	13 597
	Rappel	-	99,9920
	Précision	-	99,8519
	Reads trimmés ou *splittés*	-	3 816
	Taille manquante moyenne	-	553,1
	Reads étendus	-	24
	Taille moyenne d'extension	-	31,1
	Reads de mauvaise qualité	-	1
	Reads trop courts	-	412
	Ratio taille homopolymers	-	1,0000
	Temps	26 min 37 sec	

TABLE 4.3 – **Impact du paramètre ℓ (déterminant la longueur minimale des fragments à considérer) sur les résultats produits par ELECTOR.** Les expériences ont été réalisées sur le jeu de données *E. coli* de la Table 4.4, après correction avec HALC.

Jeu de données	*A. baylyi*	*E. coli*	*S. cerevisiae*	*C. elegans*
Génome de référence				
Souche	ADP1	K-12 substr. MG1655	W303	Bristol N2
Séquence de référence	CR543861	NC_000913	scf7180000000{084-13}	GCA_000002985.3
Taille du génome (Mbp)	3,6	4,6	12,2	100
***Reads* simulés PacBio**				
Nombre de *reads*	8 765	11 306	30 132	244 277
Longueur moyenne (bp)	8 202	8 226	8 204	8 204
Nombre de bases (Mbp)	72	93	247	2 004
Profondeur	20x	20x	20x	20x
Taux d'erreurs (%)	18,6	18,6	18,6	18,6
***Reads* Illumina**				
Numéro d'accession	ERR788913[1]	Genoscope[2]	Genoscope[3]	Simulés avec ART
Nombre de *reads*	900 000	775 500	2 500 000	20 057 100
Longueur des *reads* (bp)	250	300	250	250
Nombre de bases (Mbp)	224	232	625	5 000
Prodonfeur	50x	50x	50x	50x

TABLE 4.4 – **Description des jeux de données simulées utilisés pour les comparaisons entre ELECTOR et LRCstats.**
[1] Seul un sous-ensemble des données a été utilisé.
[2] http://www.genoscope.cns.fr/externe/nas/datasets/Illumina/ecoli/
[3] http://www.genoscope.cns.fr/externe/nas/datasets/Illumina/yeast/

de données ont été simulés à l'aide de SimLoRD, en affectant aux *reads* un taux d'erreurs de 18,6%, une profondeur de séquençage de 20x, et en laissant les autres paramètres à leurs valeurs par défaut. Les détails de ces jeux de données sont présentés Table 4.4.

Pour effectuer cette comparaison, les *reads* des différents jeux de données décrits ci-dessus ont été corrigés avec HALC, en utilisant les paramètres par défaut. Les évaluations ont ensuite été réalisées à partir des *reads* corrigés *splittés* produits par HALC. Le but visé ici étant de comparer les métriques produites par ELECTOR et par LRCstats, et non d'évaluer quel outil de correction permet d'obtenir les meilleurs résultats sur ces jeux de données, les autres outils de correction ne sont pas considérés. Un *benchmark* plus complet, présentant les résultats produits par ELECTOR pour l'ensemble des méthodes de correction décrites dans le Chapitre 3, sur divers jeux de données, est proposé dans le Chapitre 7.

La comparaison des métriques produites par ELECTOR et par LRCstats est présentée Table 4.5. Cette table met en évidence plusieurs différences de comportement entre les deux outils.

Tout d'abord, comme le soulignent les différences entre les métriques décrivant le nombre de bases des *reads* non corrigés, ELECTOR et LRCstats, n'évaluent pas les mêmes sous-ensembles de *reads*. De ce fait, les métriques décrivant les taux d'erreurs des *reads* non corrigés varient également entre ELECTOR et LRCstats. Ces différences sont liées au fait que ces deux outils reposent sur des règles différentes permettant d'exclure les *reads* considérés comme étant de trop mauvaise qualité, et donc trop difficiles à traiter lors de leurs étapes d'alignement respectives.

De plus, ELECTOR et LRCstats n'appliquent pas la même stratégie pour l'évaluation des *reads* corrigés *splittés*, et n'alignent donc pas ces derniers de la même manière. En effet, LRCstats concatène l'ensemble des différents fragments provenant d'un même *read* corrigé ayant été *splitté* par la correction, et aligne alors directement la séquence obtenue par concaténation, même si de larges régions du génome peuvent manquer entre deux fragments. Ce comportement peut grandement compliquer l'étape d'alignement, devant alors prendre en compte de larges délétions entre les séquences des *reads* corrigés construits par concaténation et les séquences

Jeu de données	Métrique	*Reads* non corrigés		*Reads* corrigés	
		ELECTOR	LRCstats	ELECTOR	LRCstats
A. baylyi	Nombre de bases	71 431 161	70 819 854	63 708 698	63 742 049
	Taux d'erreurs (%)	14,1222	17,7109	0,1113	0,0388
	Délétions	1 586 569	2 723 020	59 157	8 542
	Insertions	8 675 634	9 803 996	8 273	11 713
	Substitutions	1 224 097	499 392	5 206	8 317
	Rappel (%)	-	-	99,9937	✖
	Précision (%)	-	-	99,8909	✖
	Reads trimmés ou *splittés*	-	-	2 358	✖
	Longueur manquante moyenne (bp)	-	-	341,7	✖
	Reads étendus	-	-	24	✖
	Longueur moyenne d'extension (bp)	-	-	25,2	✖
	Reads de mauvaise qualité	-	-	2	✖
	Reads trop courts	-	-	101	✖
	Ratio longueur homopolymers	-	-	1,0000	✖
E. coli	Nombre de bases	91 950 978	90 334 058	81 199 351	81 326 352
	Taux d'erreurs (%)	14,1520	17,6204	0,1537	0,0601
	Délétions	2 036 924	3 443 967	115 973	17 460
	Insertions	11 054 008	12 456 090	15 034	21 877
	Substitutions	1 528 672	626 425	13 764	17 032
	Rappel (%)	-	-	99,9918	✖
	Précision (%)	-	-	99,8497	✖
	Reads trimmés ou *splittés*	-	-	3 816	✖
	Longueur manquante moyenne (bp)	-	-	547,3	✖
	Reads étendus	-	-	24	✖
	Longueur moyenne d'extension (bp)	-	-	31,1	✖
	Reads de mauvaise qualité	-	-	2	✖
	Reads trop courts	-	-	254	✖
	Ratio longueur homopolymers	-	-	1,0000	✖
S. cerevisiae	Nombre de bases	238 309 333	237 655 341	212 266 193	214 152 119
	Taux d'erreurs (%)	14,0333	17,5129	0,4212	0,2261
	Délétions	5 235 890	8 991 984	1 035 978	120 743
	Insertions	28 772 841	32 589 970	100 874	215 507
	Substitutions	4 058 953	16 331 23	198 853	221 646
	Rappel (%)	-	-	99,9734	✖
	Précision (%)	-	-	99,5872	✖
	Reads trimmés ou *splittés*	-	-	12 043	✖
	Longueur manquante moyenne (bp)	-	-	577,5	✖
	Reads étendus	-	-	71	✖
	Longueur moyenne d'extension (bp)	-	-	53,2	✖
	Reads de mauvaise qualité	-	-	160	✖
	Reads trop courts	-	-	3 436	✖
	Ratio longueur homopolymers	-	-	1,0580	✖
C. elegans	Nombre de bases	1 731 103 921	1 837 741 517	1 588 220 052	1 661 088 722
	Taux d'erreurs (%)	13,7683	17,1264	1,5288	0,6840
	Délétions	39 322 138	67 643 291	35 970 722	2 666 673
	Insertions	215 891 547	247 128 032	2 520 363	5 498 619
	Substitutions	30 451 721	12 106 851	2 919 936	4 888 754
	Rappel (%)	-	-	99,8945	✖
	Précision (%)	-	-	98,4961	✖
	Reads trimmés ou *splittés*	-	-	153 855	✖
	Longueur manquante moyenne (bp)	-	-	777,2	✖
	Reads étendus	-	-	772	✖
	Longueur moyenne d'extension (bp)	-	-	40,9	✖
	Reads de mauvaise qualité	-	-	2 582	✖
	Reads trop courts	-	-	170 934	✖
	Ratio longueur homopolymers	-	-	0,9978	✖

TABLE 4.5 – **Comparaison des métriques produites par ELECTOR et par LRCstats.** Les évaluations ont été réalisées après correction des *reads* par HALC, en utilisant les paramètres par défaut. Les *reads* corrigés *splittés* ainsi produits ont été fournis à ELECTOR et à LRCstats. Les tirets dans les colonnes décrivant les *reads* non corrigés indiquent que les métriques correspondantes ne sont pas calculées. Les croix dans les colonnes décrivant les résultats de LRCstats indiquent que ce dernier ne produit pas les métriques correspondantes.

des *reads* non corrigés et du génome de référence. ELECTOR, au contraire, traite les différents fragments d'un *read* corrigé *splitté* séparément avant de reconstruire l'alignement multiple global. L'étape d'alignement est ainsi facilitée, en évitant d'avoir à prendre en compte de larges régions manquantes au sein d'une unique séquence composée de plusieurs fragments. De plus, la longueur moyenne de l'ensemble des régions manquantes au sein des *reads* corrigés est également reportée par ELECTOR, mais n'est pas mentionnée par LRCstats. Ces deux approches de traitement des *reads* splittés impactent donc les résultats produits, et expliquent ainsi les différences observées sur les métriques calculées pour les *reads* corrigés. Comme souligné par ELECTOR, les *reads trimmés* et *splittés* sont nombreux après correction des données par HALC. De ce fait, les métriques reportées par ELECTOR et par LRCstats peuvent ainsi grandement varier sur les expériences présentées ici.

Enfin, les différentes stratégies d'alignement employées par les deux outils ont également un impact sur les métriques calculées. En effet, LRCstats utilise, comme base de son alignement de trois séquences, l'alignement entre le *read* non corrigé et le génome de référence, fourni par le fichier SAM produit durant la simulation des *reads*. Ce fichier étant généré séquentiellement durant la simulation, il peut contenir des évènement de type «1I1D» (*i.e.* une insertion directement suivie par une délétion), qui sont alors considérés comme tel par LRCstats. À l'inverse, les évènements de ce type sont détectés comme de simples substitutions par ELECTOR. En effet, la stratégie d'alignement multiple implémentée se base uniquement sur les séquences des différentes versions des *reads*, et non sur les fichiers produits durant la simulation, et décrivant les alignements entre le génome de référence et les *reads* simulés. Ces différences expliquent notamment les importants écarts entre les comptes d'erreurs reportés par les deux outils.

4.5.3.2 Comparaison des performances

Les temps d'exécution d'ELECTOR et de LRCstats sont tout d'abord évalués sur les jeux de données décrits Table 4.4, et ayant été utilisés afin de comparer les métriques produites par ces deux outils Section 4.5.3.1. Ces jeux de données ont été choisis afin de faire varier la taille et la complexité des génomes étudiés, ainsi que le nombre de *reads* à évaluer. Les temps d'exécution d'ELECTOR et de LRCstats sont également comparés au temps d'exécution de HALC, afin d'évaluer le coût de l'évaluation par rapport au coût de la correction. Ces résultats sont présentés Table 4.6. Pour faire varier davantage les différents facteurs pouvant affecter le temps d'exécution, mais également la consommation de mémoire, les trois jeux de données du génome d'*E. coli*, utilisés pour valider la stratégie de segmentation Section 4.5.1, sont de nouveau utilisés ici. Un jeu de données additionnel a également été simulé avec SimLoRD, en affectant à nouveau aux *reads* un taux d'erreurs de 10% et une profondeur de séquençage de 100x, mais en fixant cette fois leur longueur à un million de paires de bases. Ce nouveau jeu de de données est principalement utilisé afin de démontrer la capacité d'ELECTOR à efficacement passer à l'échelle sur des *reads* extrêmement longs. Les temps d'exécution et les consommations de mémoire de LRCstats et d'ELECTOR, sur ces quatre jeux de données, sont présentés Table 4.8.

Ces résultats montrent que le temps d'exécution et la consommation de mémoire de LRCstats augmentent grandement en fonction de la longueur des *reads*. De ce fait, LRCstats n'a pas pu être utilisé pour évaluer la correction des jeux de données composés de *reads* de 100 000 et de 1 000 000 de paires de bases, même sur un nœud de calcul disposant de 250 Go de RAM. Ce comportement, aussi bien au niveau du temps d'exécution que de la consommation de mémoire, pourrait ainsi rapidement

	A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
HALC	22 min	24 min	1 h 19 min	5 h 29 min
LRCstats	3 h 50 min	4 h 58 min	10 h 56 min	83 h 29 min
ELECTOR	14 min	28 min	1 h 55 min	32 h 55 min

TABLE 4.6 – **Comparaison des temps d'exécution d'ELECTOR et de LRCstats par rapport à la correction par HALC.** HALC a été lancé sur 16 threads, en utilisant ses paramètres par défaut. ELECTOR et LRCstats ont été lancés sur 9 threads.

devenir un problème lors de l'évaluation de jeux de données contenant de très longs *reads*.

À l'inverse, les résultats de ces expériences montrent qu'ELECTOR, grâce à sa stratégie de segmentation de l'alignement multiple, est capable d'évaluer la correction de jeux de données composés de *reads* atteignant jusqu'à un million de paires de bases, en consommant moins de 25 Go de RAM. De plus, le temps d'exécution et la consommation de mémoire d'ELECTOR n'augmentent que faiblement en fonction de la longueur des *reads*, par rapport à LRCstats. Ces observations soulignent donc la capacité d'ELECTOR à passer efficacement à l'échelle sur des *reads* extrêmement longs. Au vu de la démocratisation de tels *reads*, notamment grâce aux librairies *ultra-long reads* ONT, cette capacité de passage à l'échelle souligne un des principaux avantages d'ELECTOR face à LRCstats.

Les résultats présentés Table 4.6 montrent que LRCstats affiche des temps d'exécution 8 à 15 fois plus importants que les méthodes de correction évaluées. Un tel comportement n'est pas désirable, les méthodes d'évaluation étant vouées à fournir rapidement un descriptif de la qualité de la correction. ELECTOR présente quant à lui, des temps d'exécution grandement réduits par rapport à LRCstats. En effet, sur les expériences de la Table 4.6, ELECTOR s'est montré 2,5 à 22 fois plus rapide que LRCstats. En prenant également en compte les expériences de la Table 4.8, ELECTOR s'est même montré jusqu'à 146 fois moins coûteux que LRCstats, en termes de temps CPU. Les expériences de ces deux tables montrent ainsi que les temps d'exécution d'ELECTOR demeurent relativement comparables aux temps d'exécution des méthodes de correction évaluées. Sur les jeux de données *S. cerevisiae* et *C. elegans*, ELECTOR a cependant affiché des temps d'exécution plus élevés que HALC. Ces écarts peuvent s'expliquer, d'une part, par le fait que HALC, ELECTOR et LRCstats n'aient pas été lancés sur le même nombre de threads. D'autre part, ces écarts peuvent également être expliqués par le fait que HALC a tendance à *splitter* une large proprotion des *reads* corrigés, comme le soulignent les expériences présentées Table 4.5. Ces nombreux *reads splittés* impliquent en effet de devoir calculer de nombreux alignement multiples supplémentaires, et ont donc un impact non négligeable sur le temps d'exécution. Les temps d'exécution du module de correction de Canu, d'ELECTOR et LRCstats, sur ces mêmes jeux de données, sont également présentés Table 4.7. Sur ces expériences, Canu n'a produit que peu de *reads splittés*, comme le soulignent les métriques présentées Table 4.9. Les temps d'exécution d'ELECTOR se sont alors montrés plus faibles que ceux de Canu. Ces résultats soulignent ainsi l'un des autres avantages d'ELECTOR face à LRCstats. En effet, grâce à ses temps d'exécution réduits, ELECTOR peut aisément être utilisé afin de réduire le temps nécessaire aux analyses, en particulier lors de l'établissement de *benchmarks* comparatifs de nombreuses méthodes de correction.

	A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Canu	12 min	14 min	32 min	4 h 37 min
LRCstats	3 h 10 min	4 h 05 min	10 h 46 min	85 h 08 min
ELECTOR	8 min	11 min	30 min	4 h 19 min

TABLE 4.7 – **Comparaison des temps d'exécution d'ELECTOR et de LRCstats par rapport à la correction par Canu.** Canu a été lancé sur 16 threads, en utilisant ses paramètres par défaut. ELECTOR et LRCstats ont été lancés sur 9 threads.

Outil	Longueur des *reads* (bp)	Mémoire (Mo)	Temps	Temps CPU
LRCstats	1 000	1 803	42 min	8 h 18 min
ELECTOR	1 000	1 030	12 min	28 min
LRCstats	10 000	13 484	4 h 51 min	2 jours 22 h 38 min
ELECTOR	10 000	3 091	13 min	29 min
ELECTOR	100 000	12 231	28 min	1 h 11 min
ELECTOR	1 000 000	24 881	2 h 44 min	11 h 05 min

TABLE 4.8 – **Comparaison du temps d'exécution et de la consommation de mémoire d'ELECTOR et de LRCstats.** Les différents jeux de données ont été simulés à partir du génome d'*E. coli*, avec SimLoRD, en affectant aux *reads* un taux d'erreurs de 10%, une profondeur de séquençage de 100x, et des longueurs fixées aux valeurs indiquées. Les *reads* ont été corrigés avec le module de correction de Canu, en utilisant les paramètres par défaut. ELECTOR et LRCstats ont été lancés sur 20 threads, sur un nœud de calcul disposant de 250 Go de RAM. LRCstats n'a pas pu être lancé sur les jeux de données composés de *reads* de 100 000 et de 1 000 000 de paires de bases, dû à une consommation de mémoire supérieure à 250 Go.

4.5.4 Évaluation de données réelles

4.5.4.1 Validation des métriques en mode données réelles

Afin de valider le comportement d'ELECTOR lors de l'évaluation de données réelles, nous avons repris les jeux de données simulées présentés Table 4.4, pour comparer les résultats obtenus en mode données simulées aux résultats obtenus en mode données réelles. L'évaluation de chacun de ces jeux de données a ainsi été réalisée deux fois par ELECTOR, après correction des *reads* avec le module de correction de Canu, en utilisant les paramètres par défaut. En mode données simulées, l'ensemble des fichiers produits lors de la simulation a été fourni en entrée à ELECTOR, et les *reads* de référence ont alors été retrouvés en analysant les fichiers décrivant les erreurs introduites dans les *reads* lors de la simulation. En mode données réelles, seul le fichier FASTA décrivant les *reads* simulés a été fourni en entrée à ELECTOR, et les *reads* de référence ont alors été retrouvés en alignant les *reads* simulés sur le génome de référence à l'aide de Minimap2, comme si ces *reads* provenaient d'une réelle expérience de séquençage.

Les résultats de ces expériences sont décrits Table 4.9, et montrent que les métriques calculées par ELECTOR en mode données simulées et en mode données réelles sont cohérentes et comparables. En particulier, le rappel, la précision, et le taux d'erreurs sont très similaires entre les deux modes. Cependant, la même tendance que lors de la comparaison à LRCstats est observée, et les deux modes d'évaluation présentent ainsi des différences au niveau des comptes détaillés de chaque type d'erreur, ainsi que de plus importantes différences au niveau des *reads* reportés comme étant *trimmés* ou *splittés*, et de la longueur des régions manquantes dans ces

reads.

Ces différences peuvent s'expliquer par les biais induits par l'étape d'alignement additionnelle employée dans le mode données réelles, afin de retrouver les *reads* de référence. Les principales différences observées entre les deux modes se situent en effet dans les métriques citées ci-dessus, hautement dépendantes des choix réalisés lors de l'alignement. Par ailleurs, les métriques sur les extrémités des *reads* ayant été clippées lors de cette étape d'alignement, non calculées en mode données réelles, mais calculées en mode données simulées, expliquent également les différences observées entre les métriques des deux modes.

4.5.4.2 Résultats sur un jeu de données du génome humain

Nous présentons ici les résultats fournis par ELECTOR pour l'évaluation de la correction d'un jeu de données du génome humain composé de *reads* ONT, afin de démontrer sa capacité à s'adapter à des scénarios réalistes d'analyse de génomes longs et complexes. Le jeu de données évalué est présenté Table 4.10. Les *reads* ont été corrigés avec MECAT, en utilisant les paramètres par défaut, avant l'évaluation. Les résultats reportés par ELECTOR, ainsi que son temps d'exécution, sont présentés Table 4.11.

En utilisant 20 threads, ELECTOR a permis d'évaluer la qualité de la correction produite par MECAT en moins de 19 heures. Les résultats fournis par ELECTOR montrent que MECAT est capable de corriger des *reads* humains affichant un taux d'erreurs de près de 20%, et une profondeur de séquençage de moins de 30x, avec plus de 90% de rappel et de précision. Ces observations sont cohérentes avec les résultats présentés dans la publication de MECAT. Cette expérience souligne ainsi la capacité d'ELECTOR a être utilisé pour l'évaluation de la correction de données réelles.

4.5.5 Alignement et assemblage

Les résultats du second module d'ELECTOR, sur les jeux de données simulées présentés Table 4.4, et après correction des *reads* avec HALC, en utilisant les paramètres par défaut, sont présentés Table 4.12. Ces résultats ont été obtenus à partir des *reads* corrigés *splittés* produits par HALC.

Les métriques obtenues par ce second module, et plus particulièrement celles décrivant la qualité d'alignement des *reads* corrigés sur le génome de référence, sont cohérentes avec les métriques produites par le premier module d'ELECTOR. Ces résultats penchent également du côté d'ELECTOR concernant les métriques dissimilaires au sein des expériences de comparaison avec LRCstats, présentées Table 4.5. En particulier, sur le jeu de données *C. elegans*, le premier module d'ELECTOR reporte un taux d'erreurs de 1,5288%, tandis que LRCstats reporte un taux d'erreurs de 0,6840% pour les *reads corrigés*. Pour ce même jeu de données, le second module d'ELECTOR, calculant les alignements des *reads* corrigés sur le génome de référence à l'aide de Minimap2, reporte une identité d'alignement moyenne de 98,5405%. Cette valeur est en effet davantage compatible avec le taux d'erreurs reporté par le premier module d'ELECTOR qu'avec le taux d'erreurs reportés par LRCstats.

Les métriques décrivant la qualité de l'assemblage permettent également de mettre en avant davantage de détails concernant la qualité des *reads* corrigés, ne pouvant être abordés par le premier module d'ELECTOR. En effet, le simple fait que ce module reporte des valeurs satisfaisantes pour le taux d'erreurs, le rappel, la précision,

Jeu de données	Métrique	*Reads* non corrigés		*Reads* corrigés	
		Mode simulé	Mode réel	Mode simulé	Mode réel
A. baylyi	Nombre de bases	70 986 268	70 971 457	66 632 351	66 617 540
	Taux d'erreurs (%)	14,2083	14,3458	4,9496	5,0507
	Délétions	1 595 606	1 669 513	738 704	818 300
	Insertions	8 714 975	8 276 543	3 517 266	3 170 305
	Substitutions	1 248 229	1 169 783	622 602	553 027
	Rappel (%)	-	-	95,2430	95,1207
	Précision (%)	-	-	95,0573	94,9563
	Reads trimmés ou *splittés*	-	-	547	1 338
	Longueur manquante moyenne (bp)	-	-	27,1	74,8
	Reads étendus	-	-	53	56
	Longueur moyenne d'extension (bp)	-	-	32,3	32,5
	Reads de mauvaise qualité	-	-	0	0
	Reads trop courts	-	-	0	0
	Ratio longueur homopolymers	-	-	1,0055	1,0055
E. coli	Nombre de bases	91 933 413	91 933 413	86 443 218	86 443 218
	Taux d'erreurs (%)	14,3159	14,4801	5,2422	5,3805
	Délétions	2 102 399	2 216 769	1 011 985	1 135 461
	Insertions	11 365 447	10 746 089	4 804 729	4 309 866
	Substitutions	1 615 193	1 505 319	841 679	744 167
	Rappel (%)	-	-	94,9490	94,7901
	Précision (%)	-	-	94,7647	94,6263
	Reads trimmés ou *splittés*	-	-	764	1 843
	Longueur manquante moyenne (bp)	-	-	27,2	70,4
	Reads étendus	-	-	66	58
	Longueur moyenne d'extension (bp)	-	-	27,4	28,8
	Reads de mauvaise qualité	-	-	0	0
	Reads trop courts	-	-	0	0
	Ratio longueur homopolymers	-	-	1,0000	1,0000
S. cerevisiae	Nombre de bases	244 560 743	244 402 568	229 555 492	229 403 697
	Taux d'erreurs (%)	14,2545	14,4200	5,0619	5,2003
	Délétions	5 483 119	5 800 286	2 574 320	2 916 564
	Insertions	30 090 583	28 452 967	12 252 413	10 965 458
	Substitutions	4 375 017	4 081 445	2 197 172	1 940 888
	Rappel (%)	-	-	95,1477	94,9880
	Précision (%)	-	-	94,9459	94,8074
	Reads trimmés ou *splittés*	-	-	2 216	4 943
	Longueur manquante moyenne (bp)	-	-	35,1	74,7
	Reads étendus	-	-	178	169
	Longueur moyenne d'extension (bp)	-	-	30,7	31,9
	Reads de mauvaise qualité	-	-	43	42
	Reads trop courts	-	-	0	0
	Ratio longueur homopolymers	-	-	0,9937	0,9937
C. elegans	Nombre de bases	1 997 798 872	1 997 329 137	1 873 188 109	1 872 722 093
	Taux d'erreurs (%)	14,2697	14,4219	4,9561	5,0691
	Délétions	44 146 540	46 582 049	20 233 048	22 866 866
	Insertions	245 675 065	232 756 251	97 517 920	87 347 016
	Substitutions	36 862 525	34 467 779	17 594 481	15 522 682
	Rappel (%)	-	-	95,2663	95,1333
	Précision (%)	-	-	95,0524	94,9394
	Reads trimmés ou *splittés*	-	-	14 407	36 524
	Longueur manquante moyenne (bp)	-	-	29,5	71,0
	Reads étendus	-	-	1 462	1 372
	Longueur moyenne d'extension (bp)	-	-	32,5	33,4
	Reads de mauvaise qualité	-	-	305	301
	Reads trop courts	-	-	0	0
	Ratio longueur homopolymers	-	-	1,0120	1,0120

TABLE 4.9 – **Comparaison des métriques produites par ELECTOR en mode données simulées et en mode données réelles.** Les évaluations ont été réalisées après correction des *reads* par le module de correction de Canu, en utilisant les paramètres par défaut.

Jeu de données	*H. sapiens*
Génome de référence	
Souche	GRCh30[1]
Séquence de référence	NC_000001.11
Taille du génome (Mbp)	249
***Reads* ONT réels**	
Nombre de *reads*	1 075 867
Longueur moyenne (bp)	6 828
Nombre de bases (Mbp)	7 120
Profondeur	29x
Taux d'erreurs (%)	17,60
Numéro d'accession	PRJEB23027[2]

TABLE 4.10 – **Description du jeu de données du génome humain utilisé pour évaluer ELECTOR sur des données réelles.** [1] Seul le chromosome 1 a été utilisé. [2] Seuls les *reads* du chromosome 1 ont été utilisés.

	Reads non corrigés	*Reads corrigés*
Nombre de bases	5 605 157 590	5 451 767 836
Taux d'erreurs (%)	19,74	8,61
Insertions	247 953 086	10 144 736
Délétions	746 165 024	473 239 036
Substitutions	162 822 923	7 521 389
Rappel (%)	-	92,70
Précision (%)	-	91,50
Reads trimmés ou *splittés*	-	570 635
Longueur manquante moyenne (bp)	-	362,0
Reads étendus	-	275
Longueur moyenne d'extension (bp)	-	62,4
Reads de mauvaise qualité	-	356
Reads trop courts	-	4 279
Ratio longueur homopolymers	-	0,7570
Temps	-	18 h 27 min

TABLE 4.11 – **Résultats produits par ELECTOR pour l'évaluation d'un jeu de données réelles du génome humain.** Les *reads* ont été corrigés avec MECAT, en utilisant les paramètres par défaut, avant l'évaluation. ELECTOR a été lancé sur 20 threads.

	A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Alignement				
Nombre de *reads*	9 103	12 461	38 926	580 615
Nombre de bases	63 752 074	81 338 740	214 233 733	1 662 759 783
Longueur moyenne (bp)	7 003	6 527	5 503	2 863
Reads alignés (%)	100,0000	100,0000	99,7971	98,7408
Identité moyenne (%)	99,9329	99,9387	99,4886	98,5405
Couverture du génome (%)	99,9523	99,9754	98,9989	99,9659
Assemblage				
Nombre de contigs	16	26	168	2 147
Nombre de contigs alignés	16	26	168	2 147
Nombre de *breakpoints*	0	1	5	71
NGA50 (bp)	305 257	262 928	94 358	14 726
NGA75 (bp)	204 349	132 928	55 321	1 615
Couverture du génome (%)	99,1989	99,1540	91,7579	57,0561
Temps	1 min	1 min	3 min	29 min
Mémoire (Mo)	848	968	2 824	19 173

TABLE 4.12 – **Résultats d'alignement et d'assemblage produits par le second module d'ELECTOR.** Les *reads* ont été corrigés avec HALC, en utilisant les paramètres par défaut. Les évaluations ont alors été réalisées à partir des *reads* corrigés *splittés* ainsi produits. Les alignements et les assemblages ont été lancés sur 9 threads.

et que peu de *reads* soient considérés comme trop courts ou de trop mauvaise qualité, ne signifie pas forcément que l'assemblage généré à partir ces *reads* atteindra une qualité satisfaisante. Par exemple, pour le jeu de données *A. baylyi*, le premier module d'ELECTOR reporte un rappel de 99,9937%, une précision de 99,8909%, un taux d'erreurs de 0,1113% et uniquement 2 *reads* de trop mauvaise qualité et 101 *reads* trop courts. Cependant, l'assemblage généré à partir de ces *reads* corrigés comporte 16 contigs, contre un unique chromosome dans le génome de référence. L'assemblage affiche également un NGA50 de 305 000 paires de bases, contre un génome de référence de 3,6 millions de paires de bases, et ne représente donc pas un assemblage de bonne qualité. La qualité de cet assemblage est cependant cohérente avec la tendance de HALC à hautement *splitter* les *reads*, résultant ainsi en de multiples fragments corrigés de haute qualité, mais courts, et alors plus difficiles à assembler.

4.6 Synthèse

Dans ce chapitre, nous avons présenté ELECTOR, un nouvel outil permettant d'évaluer les méthodes de correction de *reads* longs. ELECTOR produit une large variété de métriques, en comparant base à base, via un alignement multiple, les trois différentes versions d'un même *read* : le *read* original non corrigé, le *read* corrigé, et le *read* de référence, correspondant à la portion du génome de référence d'où provient le *read* original, et ne contenant aucune erreur. Ces métriques incluent notamment le rappel, la précision, ainsi que les taux d'erreurs et les comptes détaillés d'insertions, de délétions, et de substitutions, pour les *reads* corrigés et pour les *reads* non corrigés. Le rappel et la précision, en particulier, permettent d'analyser précisément les comportements problématiques des outils de correction, en permettant notamment de détecter les potentielles erreurs additionnelles introduites par la correction. De

tels comportements demeurent en effet indétectables en analysant uniquement les taux d'erreurs des *reads* avant et après la correction, comme le propose LRCstats, le seul outil permettant d'évaluer les méthodes de correction de *reads* longs disponible jusqu'alors. De plus, ELECTOR informe également à propos des difficultés des méthodes de correction à traiter certaines régions des *reads* longs, en reportant le nombre de *reads trimmés* ou *splittés*. Par ailleurs, ELECTOR permet également d'évaluer la qualité d'alignement des *reads* longs corrigés sur le génome de référence, ainsi que la qualité des assemblages pouvant être générés à partir des *reads* longs corrigés, apportant ainsi de nouvelles informations sur la qualité de la correction, non fournies par LRCstats. ELECTOR présente ses résultats sous forme d'un résumé au format texte et au format PDF, mais également sous forme de graphiques illustrant les différentes métriques.

ELECTOR a été développé comme un outil rapide, et capable de passer efficacement à l'échelle, afin de répondre à des problématiques réalistes d'application des *reads* longs à des génomes longs et complexes, et afin de pouvoir s'adapter aux longueurs continuellement croissantes des *reads* longs. Pour cela, la stratégie d'alignement multiple de triplets de *reads* longs a été adaptée à cet objectif de passage à l'échelle. Une heuristique de segmentation pour l'alignement multiple de séquences longues et bruitées a ainsi été développée. Cette stratégie de segmentation réduit grandement le temps d'exécution d'ELECTOR, étant jusqu'à 171 fois plus rapide que l'approche d'alignement multiple classique, et permet ainsi de répondre aux problématiques sus-citées. Nous montrerons, dans le Chapitre 6, comment cette stratégie de segmentation peut être généralisée au calcul d'alignements multiples de plus grands ensembles de *reads* longs.

De nombreuses expériences ont été réalisées sur des jeux de données provenant de génomes de taille et de complexité variables, de la bactérie *A. baylyi* à l'humain, et sur des *reads* de longueur croissante, de 1 000 jusqu'à 1 000 000 de paires de bases. Ces expériences ont permis de démontrer l'efficacité d'ELECTOR par rapport à LRCstats en termes de temps d'exécution, ce dernier étant 2,5 à 22 fois fois plus lent qu'ELECTOR, et jusqu'à 146 fois plus coûteux en termes de temps CPU, sur l'ensemble des jeux de données évalués. Cette importante réduction des temps d'exécution souligne ainsi l'intérêt d'ELECTOR lors de l'établissement de *benchmarks* comparatifs de grands nombres d'outils de correction. De plus, ces expériences ont également permis de démontrer la capacité d'ELECTOR à passer efficacement à l'échelle sur des *reads* de longueur extrêmement importante, aussi bien en termes de temps d'exécution qu'en termes de consommation de mémoire. Au contraire, LRCstats, n'a pas pu être lancé sur des *reads* plus longs que 10 000 paires de bases, à cause d'une consommation de mémoire déraisonnable de plus de 250 Go. Ces résultats soulignent ainsi l'intérêt d'ELECTOR lors de l'évaluation de *reads* très longs, tels que les *reads* provenant de librairies *ultra-long reads* ONT. En effet, nos expériences ont également permis de confirmer qu'ELECTOR, bien qu'initialement développé pour l'évaluation de données simulées, est également compatible avec des données réelles, et produit des résultats cohérents dans les deux cas.

ELECTOR semble ainsi être une base intéressante pour la réalisation de *benchmarks* incluant de nombreux outils de corrections, utiles aussi bien pour les utilisateurs cherchant à déterminer quel correcteur est le plus adapté à leurs données, que pour les développeurs cherchant à comparer leurs outils aux méthodes de l'état de l'art. Cet aspect est d'autant plus renforcé par le fait qu'ELECTOR est facile d'utilisation, et ne demande aucun pré-traitement à l'utilisateur. Nous présenterons ainsi, dans le Chapitre 7, un *benchmark* réalisé à l'aide d'ELECTOR sur l'ensemble des méthodes de correction présentées dans le Chapitre 3, ainsi que sur les deux nouvelles

méthodes de correction introduites dans le Chapitre 5 et dans le Chapitre 6, sur divers jeux de données.

Chapitre 5

Correction hybride de *reads* longs fortement bruités

Sommaire

5.1 Introduction

5.1.1 Contexte

La technologie de séquençage ONT est disponible en accès anticipé depuis 2014, et est commercialisée depuis 2015. Cependant, depuis l'apparition de cette technologie, la plupart des travaux introduisant de nouveaux outils de correction continuent à valider ces derniers uniquement sur des données PacBio. En effet, comme montré dans le Chapitre 3, Table 3.1 et Table 3.2, sur les 19 méthodes de correction de *reads* longs développées après 2014, seules 7 d'entre elles ont été validées sur des *reads* ONT dans leurs publications respectives.

Bien que permettant d'attester de la qualité globale des méthodes de correction, les taux d'erreurs des *reads* PacBio, compris entre 10 et 15% en moyenne, sont bien moins élevés que les taux d'erreurs des *reads* ONT. En particulier, les *reads* séquencés à l'aide des premières versions des chimies ONT, affichent des taux d'erreurs supérieurs à 15% en moyenne, et pouvant atteindre jusqu'à 30%. De plus, contrairement aux *reads* PacBio, les *reads* ONT présentent des erreurs systématiques dans les homopolymers.

Ces différences peuvent donc affecter la qualité de la correction, et limiter la capacité des correcteurs à s'adapter au traitement des *reads* ONT. En particulier, peu de méthodes, aussi bien de correction hybride que d'auto-correction, permettent une correction efficace des *reads* longs issus des premières expériences de séquençage ONT. Plus précisément, la plupart des méthodes de correction tendent à produire des *reads* longs corrigés affichant toujours d'importants taux d'erreurs, ou à ne corriger que les régions des *reads* longs disposant déjà d'une qualité relativement élevée, produisant ainsi un faible débit de bases corrigées et résultant en de larges proportions de *reads trimmés* ou *splittés*. Dans un cas comme dans l'autre, de tels résultats peuvent également limiter les performances des assembleurs, et résulter ainsi en des assemblages insatisfaisants, composés de nombreux contigs courts, et ne couvrant pas l'intégralité des génomes originaux.

Bien que les technologies de séquençage de troisième génération évoluent rapidement, et que les taux d'erreurs des *reads* longs, aussi bien pour la technologie PacBio et que pour la technologie ONT, atteignent désormais 10 à 12% en moyenne, le reséquençage reste coûteux et n'est donc pas accessible à l'intégralité des laboratoires. Ainsi, parvenir à corriger les données issues des premières expériences de séquençage ONT demeure intéressant, en plus d'être un challenge au niveau algorithmique.

5.1.2 Travaux précédents

Comme présenté dans le Chapitre 3, décrivant l'état de l'art des méthodes de correction de *reads* longs, la correction hybride se divise en quatre approches : l'alignement de *reads* courts, l'alignement de contigs issus d'un assemblage de *reads* courts, l'utilisation de graphes de de Bruijn, et l'utilisation de modèles de Markov cachés. Chacune de ces approches présente cependant un certain nombre d'inconvénients.

Les méthodes reposant sur l'alignement, aussi bien des *reads* courts que des contigs obtenus après assemblage de ces derniers, permettent de corriger de façon satisfaisante les régions des *reads* longs sur lesquelles des alignements peuvent être mis en évidence. Cependant, il reste toutefois difficile d'aligner les données issues de *reads* courts sur les régions très bruitées des *reads* longs. Ces méthodes tendent alors à *trimmer* ou à *splitter* de larges proportions des *reads* longs, et donc à produire

de nombreux fragments de bonne qualité, mais courts, perdant ainsi l'information portée par l'importante longueur des *reads*.

L'approche par modèles de Markov cachés, quant à elle, repose également sur une première étape d'alignement des *reads* courts sur les *reads* longs, et souffre donc également des limitations décrites ci-dessus.

Enfin, pour les méthodes reposant sur des graphes de de Bruijn, trouver des ancres valides sur les *reads* longs, permettant de placer ces derniers de façon correcte sur le graphe est parfois difficile, notamment lorsque ces ancres sont déterminées à partir des *k*-mers solides des *reads*. En particulier, lorsque le taux d'erreurs des *reads* longs est très élevé, ou que le génome étudié est complexe et contient de nombreuses régions répétées, la simple étude de la fréquence des *k*-mers ne constitue pas une approche pertinente. En effet, de nombreux *k*-mers contenant des erreurs peuvent alors apparaître plus fréquemment que le seuil de solidité fixé, et être utilisés comme ancres, malgré la présence de ces erreurs. De telles ancres peuvent alors empêcher la mise en évidence de chemins, ces derniers étant recherchés à partir de points de départ et / ou d'arrivée incorrects, ou même introduire des erreurs additionnelles dans la correction, diminuant ainsi la qualité des *reads* longs corrigés produits.

Parmi toutes les méthodes présentées dans l'état de l'art, NaS, développée spécialement pour la correction de *reads* longs ONT, présente une approche innovante et intéressante, reposant sur l'alignement, mais permettant de s'affranchir des biais liés aux régions non couvertes des *reads* longs. En effet, NaS aligne tout d'abord les *reads* courts sur les *reads* longs, afin de permettre la correction des régions couvertes par les alignements. Un processus de recrutement de nouveaux *reads* courts est ensuite proposé, en comparant les *reads* courts alignés aux autres *reads* courts, afin d'en recruter de nouveaux, et ainsi permettre de couvrir les régions des *reads* longs non couvertes par les alignements initiaux. Les ensembles de *reads* courts ainsi obtenus sont alors assemblés, afin de fournir une correction aux *reads* longs originaux auxquels ils sont associés.

CoLoRMap, avec son approche par OEA (*One-End Anchors*, voir Section 3.2.1.7), propose également une stratégie permettant de lever les biais liés aux régions non couvertes des *reads* longs, en recrutant et en assemblant de nouveaux *reads* courts. Cependant, CoLoRMap s'appuie sur les informations portées par les *reads* pairés, et nécessite donc qu'un des *reads* d'une paire donnée soit aligné sur une des régions bordant la région non couverte, afin de recruter l'autre *read* de la paire. Un tel comportement peut être problématique, notamment si la région non couverte à corriger est longue, la distance entre deux *reads* d'une paire étant limitée, et le processus de recrutement ne permettant alors pas de couvrir l'intégralité de cette région. De plus, cette approche de correction est limitée uniquement aux librairies de *reads* courts pairés, ne représentant pas l'intégralité des expériences de séquençage de *reads* courts. NaS, avec son approche de comparaison des *reads* courts alignés à l'ensemble des autres *reads* courts, permet de s'affranchir de ces limitations, et ainsi de mieux corriger les régions non couvertes par les alignements initiaux.

Cependant, NaS souffre de temps d'exécution extrêmement importants, atteignant plusieurs jours, même pour des génomes bactériens. La majorité de ce temps est due à l'étape de recrutement des *reads* courts additionnels, nécessitant la comparaison des *reads* courts déjà alignés à l'ensemble des autres *reads* courts. En effet, cette étape est responsable, en moyenne, de 70% du temps d'exécution total de NaS. Conserver la méthodologie d'alignement et de recrutement de NaS, en optimisant l'étape de recrutement, pourrait donc permettre de réduire le temps de traitement, tout en maintenant l'intérêt de NaS, notamment pour la réduction des biais liés aux régions des *reads* longs non couvertes par les alignements initiaux.

5.1.3 Contribution

Afin de répondre aux problématiques sus-citées, nous avons développé HG-CoLoR[1], une nouvelle méthode de correction hybride. Cette dernière reprend le principe de NaS, initiant la correction en alignant les *reads* courts sur les *reads* longs, et faisant usage de l'information portée par les autres *reads* courts afin de couvrir et de corriger les régions des *reads* longs sur lesquelles aucun *read* court n'a pu être aligné. Pour réduire les biais liés aux importants temps d'exécution de NaS, le processus de recrutement est cependant remplacé par l'utilisation d'un graphe de de Bruijn d'ordre variable, construit à partir des *reads* courts.

HG-CoLoR repose en effet sur une stratégie de type *seed and extend*, où les *reads* courts alignés sur un *read* long donné sont utilisés en tant que graines, et ancrés sur le graphe de de Bruijn d'ordre variable. Celui-ci est alors traversé afin de trouver un chemin permettant de relier ces graines, et ainsi corriger les régions du *read* long non couvertes par des alignements. Cette étape d'extension à l'aide du graphe permet ainsi de mimer le processus de recrutement de NaS, en évitant cependant d'avoir à comparer directement l'ensemble des *reads* courts entre eux, et en faisant plutôt usage de l'information portée par le graphe concernant les chevauchements entre les différents k-mers des *reads* courts. Ainsi, HG-CoLoR combine les avantages des méthodes reposant sur une stratégie d'alignement, permettant la correction des régions couvertes des *reads* longs, aux avantages des méthodes reposant sur l'utilisation d'un graphe de de Bruijn, ne nécessitant pas d'alignement, et permettant ainsi la correction des régions non couvertes des *reads* longs.

Certaines similarités sont donc partagées avec les méthodes de correction hybride basées sur des graphes de de Bruijn, plus particulièrement avec LoRDEC et FMLRC. En effet, comme ces deux méthodes, des paires d'ancres sont utilisées, afin de définir un point de départ et un point d'arrivée sur le graphe, et d'y rechercher des chemins permettant de relier ces ancres. Cependant, ces deux méthodes définissent leurs ancres à partir des k-mers solides des *reads* longs. Comme mentionné Section 5.1.2, définir les ancres de cette façon n'est cependant pas satisfaisant, des k-mers contenant des erreurs, mais étant relativement fréquents, pouvant être utilisés comme graines, et diminuer la qualité de la correction. HG-CoLoR, en utilisant comme ancres des k-mers provenant des *reads* courts préalablement alignés sur les *reads* longs, s'assure de la qualité des ancres utilisées, et réduit donc les biais pouvant être induits par des traversées du graphe entre des ancres contenant des erreurs.

L'idée d'utiliser un graphe de de Bruijn d'ordre variable est également commune entre HG-CoLoR et FMLRC. Cependant, tandis que FMLRC propose d'utiliser un tel graphe afin de réaliser deux passes de correction avec deux valeurs de k différentes, HG-CoLoR propose d'explorer l'ensemble des différents graphes de de Bruijn. En effet, à chaque sommet, l'ordre du graphe est localement diminué si aucun chemin ne peut être suivi à partir de celui-ci. HG-CoLoR permet ainsi d'explorer l'intégralité des graphes, lors de la recherche d'un chemin entre deux graines. De ce fait, les cas problématiques dans lesquels aucun chemin ne peut être suivi pour relier une paire de graines donnée sont grandement réduits, permettant ainsi la correction de plus larges régions des *reads* longs. De plus, HG-CoLoR est antérieur à FMLRC et a donc été le premier outil de correction hybride à introduire l'utilisation d'un graphe de de Bruijn d'ordre variable.

1. https://github.com/morispi/HG-CoLoR

5.2 Graphe de de Bruijn d'ordre variable

5.2.1 Travaux précédents

Comme mentionné Section 2.5.3, Section 3.2.3 et Section 3.3.2, les graphes de de Bruijn sont largement utilisés en bioinformatique, non seulement pour résoudre des problèmes d'assemblage, mais également, depuis plus récemment, pour résoudre des problèmes de correction de *reads* longs. Cependant, malgré l'utilité de ces graphes, certaines difficultés leurs sont étroitement liées, dû au fait que la valeur de k soit fixée dès la construction. En effet, le choix d'une grande valeur de k permet au graphe de mieux résoudre les régions répétées, mais résulte en des arcs manquants dans les régions du génome trop peu couvertes lors du séquençage. Au contraire, le choix d'une petite valeur de k permet de retrouver les arcs dans ces régions faiblement couvertes, mais résulte alors en un graphe contenant de nombreux embranchements, rendant ainsi la résolution des régions répétées plus difficile.

Pour résoudre ce problème, la plupart des assembleurs modernes utilisent généralement plusieurs graphes de de Bruijn, construits pour des ordres différents. Bien que cette approche permette effectivement d'améliorer la qualité des assemblages produits, elle impacte également grandement le temps d'exécution et la consommation de mémoire, puisque plusieurs graphes doivent alors être construits et stockés.

Ainsi, plus récemment, des méthodes permettant de représenter l'intégralité des graphes de de Bruijn, jusqu'à un ordre maximal K, au sein d'une unique structure de données, ont été développées. De telles structures sont alors qualifiées de graphes de de Bruijn d'ordre variable, comme mentionné Section 3.2.3.3. Par exemple, le *manifold de Bruijn Graph* [75] associe des séquences de longueurs arbitraires aux sommets du graphe, plutôt que d'y associer des k-mers de longueur fixée. Cette structure de données est cependant principalement d'intérêt théorique, puisqu'aucune implémentation n'a encore été proposée. Une autre implémentation des graphes de de Bruijn d'ordre variable a été proposée dans [10]. Cette implémentation repose sur la représentation succincte des graphes de de Bruijn proposée dans [11], et supporte des opérations supplémentaires permettant de modifier l'ordre du graphe à la volée, au moment de l'exécution. Cependant, l'implémentation actuelle de cette structure de données limite l'ordre maximal du graphe à $K = 64$. Une telle limitation est trop restrictive, l'ordre maximal du graphe devant être dicté par les données utilisées, notamment en fonction de leur taux d'erreurs et de leur profondeur de séquençage, plutôt que par une valeur arbitraire.

De ce fait, l'implémentation de FMLRC n'étant pas encore disponible au moment de ce travail, une nouvelle implémentation des graphes de de Bruijn d'ordre variable a été développée pour HG-CoLoR. Cette implémentation repose sur PgSA [58], une structure d'indexation permettant de stocker un ensemble de séquences, afin de répondre à une liste de requêtes.

5.2.2 PgSA

PgSA est une structure de données permettant l'indexation d'un ensemble de séquences, afin de répondre à la liste de requêtes suivante, pour une chaîne de caractères s donnée :

1. Dans quelles séquences apparaît s ?
2. Dans combien de séquences apparaît s ?
3. Quelles sont les positions des occurrences de s ?

4. Quel est le nombre d'occurrences de s ?
5. Dans quelles séquences s n'apparaît qu'une seule fois ?
6. Dans combien de séquences s n'apparaît qu'une seule fois ?
7. Quelles sont les positions des occurrences de s dans les séquences où s n'apparaît qu'une seule fois ?

Dans ces requêtes, s peut être fournie directement sous la forme d'une chaîne de caractères, ou sous la forme d'une paire d'entiers, représentant respectivement l'identifiant d'une des séquences indexées, et la position de l'occurrence de s au sein de cette séquence. Les résultats des requêtes sont, quant à eux, fournis sous des formats différents, en fonction du type de la requête. Ainsi, pour les requêtes de dénombrement (*i.e.* les requêtes 2, 4, 5, et 6), le résultat est fourni sous la forme d'un entier. Pour la première requête, le résultat est fourni sous la forme d'un ensemble d'entiers, correspondant aux identifiants des séquences au sein desquelles s apparaît. Enfin, pour les requêtes concernant les positions des occurrences (*i.e.* les requêtes 3 et 7), le résultat est fourni sous la forme d'un ensemble de paires d'entiers, chacune de ces paires représentant l'identifiant d'une séquence dans laquelle s apparaît, ainsi que la position de son occurrence au sein de cette séquence.

Comme mentionné précédemment, un index des séquences doit être construit afin de pouvoir répondre à l'ensemble de ces requêtes. L'index de PgSA est construit comme suit. Tout d'abord, toutes les séquences présentant des chevauchements sont combinées, afin de créer un *pseudogénome*. Pour éviter une procédure d'assemblage coûteuse, une procédure plus légère a été mise en place dans PgSA. Les chevauchements exacts entre les séquences sont recherchés dans l'ordre décroissant de leur longueur. Une fois un chevauchement entre deux séquences trouvé, les deux séquences sont combinées. Une séquence donnée ne peut ainsi être combinée qu'avec une unique autre séquence. Si certaines séquences n'ont pas pu être considérées, puisque ne présentant aucun chevauchement avec les autres séquences, elles sont alors concaténées à la fin du pseudogénome généré lors de l'étape précédente. Une fois toutes les séquences considérées, et le pseudogénome complet construit, la table des suffixes échantillonnée de ce pseudogénome est alors calculée. Un tableau auxiliaire, permettant de retrouver les positions des occurrences des séquences originales au sein du pseudogénome est également calculé. Chaque élément de ce tableau auxiliaire associe ainsi l'identifiant d'une séquence originale à la position correspondante de son occurrence au sein du pseudogénome. Chaque élément contient également des données additionnelles, apportant des informations complémentaires à propos de la séquence associée à l'élément (séquence répétée ou occurrence unique, par exemple), et utilisées afin de traiter les différentes requêtes. Ces dernières sont résolues à l'aide d'une recherche dichotomique dans la table des suffixes, couplée à l'utilisation des informations complémentaires contenues dans la table auxiliaire.

Puisque les séquences sont combinées en fonction de leurs chevauchements lors de la création du pseudogénome, et puisque PgSA ne conserve aucune information concernant leurs longueurs, seul un ensemble de séquences de taille constante peut être indexé. Cependant, la longueur des chaînes de caractères utilisées dans les requêtes n'est, quant à elle, pas fixée au moment de la construction. Ainsi, PgSA permet le traitement des requêtes présentées précédemment pour des chaînes de caractères s de longueur variable, sans avoir besoin de reconstruire l'index. Cette propriété se montre alors particulièrement intéressante pour la représentation des graphes de de Bruijn d'ordre variable. En effet, l'index peut être interrogé avec des suffixes de longueur variable des k-mers associés aux sommets du graphe, et l'ordre du graphe peut ainsi être modifié à la volée, lors de l'exécution.

5.2.3 Représentation

Pour représenter le graphe de de Bruijn d'ordre variable d'un ensemble de *reads* à l'aide de PgSA, un ordre maximal K est choisi. L'index est alors construit pour l'ensemble des K-mers des *reads*, afin d'être en mesure de représenter l'ensemble de tous les graphes de de Bruijn, jusqu'à cet ordre maximal. Tout graphe peut alors être parcouru, quelque soit l'ordre choisi $k \leq K$, en interrogeant l'index, et sans jamais nécessiter d'étape de construction explicite. Ainsi, pour traverser le graphe, pour l'ordre K, et à partir d'un sommet donné u, l'index est interrogé, en utilisant la troisième requête (*i.e.* quelles sont les positions des occurrences de s ?), avec s défini comme étant le suffixe de longueur $K-1$ du K-mer représenté par le sommet u. Cette requête retourne alors un ensemble de paires d'entiers, chacune de ces paires représentant l'identifiant d'un K-mer, et la position de l'occurrence de s au sein de ce K-mer.

Les paires d'entiers contenues dans cet ensemble sont alors traitées, et seules celles dont la position ne représente pas le suffixe de longueur $K-1$ du K-mer associé sont conservées, afin de pouvoir étendre ces occurrences d'un caractère sur la droite. Ces occurrences étendues représentent alors les K-mers ayant un chevauchement préfixe / suffixe de longueur $K-1$ avec le K-mer représenté par le sommet u, et permettent ainsi de définir les arcs sortants de ce sommet, et donc de traverser le graphe, pour l'ordre K.

Afin de modifier l'ordre, et de traverser le graphe avec un ordre $k \leq K$, à partir d'un sommet donné u, pour l'ordre K, l'index est de nouveau interrogé, toujours avec la troisième requête, mais en définissant cette fois s comme le suffixe de longueur $k-1$ du K-mer représenté par le sommet u. De la même façon, les paires d'entiers retournées par la requête sont traitées, et seules celles dont la position ne représente pas le suffixe de longueur $k-1$ du K-mer associé sont conservées, afin de pouvoir étendre ces occurrences d'un caractère sur la droite. Ces occurrences étendues représentent alors cette fois les k-mers ayant un chevauchement préfixe / suffixe de longueur $k-1$ avec le suffixe de longueur k du K-mer représenté par le sommet u. Elles permettent alors de définir les arcs sortants du sommet correspondant au suffixe de longueur k du K-mer associé au sommet u, et donc de traverser le graphe, pour l'ordre k.

Les arcs sortants d'un sommet donné étant calculés en interrogeant l'index, il est alors également aisé de traverser le graphe en arrière, et donc de calculer les arcs entrants d'un sommet. Pour cela, pour un ordre donné $k \leq K$, plutôt que d'interroger l'index avec les suffixes des K-mers représentés par les sommets, l'index est simplement interrogé avec leurs préfixes. Les paires de positions retournées sont alors traitées de la même façon que précédemment, à l'exception que seules les paires dont les positions ne représentent pas le préfixe de longueur $k-1$ du K-mer associé sont conservées, afin de pouvoir étendre ces occurrences d'un caractère sur la gauche, et ainsi définir les arcs entrants pour le sommet et l'ordre désiré.

L'algorithme permettant de retrouver tous les arcs, entrants ou sortants, d'un sommet, pour un ordre donné $k \leq K$, au sein d'un graphe de de Bruijn d'ordre variable représenté par PgSA est donné Figure 5.1. Le processus est également illustré Figure 5.2, pour le calcul des arcs sortants d'un sommet.

```
CALCUL DES ARCS D'UN SOMMET(s, k, K, b)
▷ Entrée : s chaîne de caractères, k entier, K entier, b booléen
▷ Sortie : E, les arcs du sommet représentant le k-mer s dans le graphe de de
▷          Bruijn d'ordre k. K désigne l'ordre maximal du graphe représenté
▷          par PgSA. Si b = 0, le graphe est traversé en avant, sinon, il est
▷          traversé en arrière.
▷ Auxiliaire : occs ensemble de paires d'entiers, i entier, id entier, pos entier
1  Début
2    E ← ∅
3    occs ← ∅
4    si b = 0 alors
5      occs ← POSITIONSDESOCCURRENCES(s[1..k − 1])
6    sinon
7      occs ← POSITIONSDESOCCURRENCES(s[0..k − 2])
8    i ← 0
9    tantque i < LONGUEUR(occs) et CARDINAL(E) < 4 faire
10     (id, pos) ← occs[i]
11     si b = 0 et pos + k ≤ K alors
12       E ← E ∪ {(s, IDVERSKMER(id)[pos..pos + k − 1])}
13     sinon si b = 1 et pos > 0 alors
14       E ← E ∪ {(s, IDVERSKMER(id)[pos − 1..pos + k − 2])}
15     i ← i + 1
16   retourner E
17 Fin
```

FIGURE 5.1 – **Algorithme de calcul des arcs d'un sommet d'un graphe de de Bruijn d'ordre variable représenté par PgSA.** POSITIONSDESOCCURRENCES et IDVERSKMER sont des fonctions de PgSA permettant respectivement de récupérer la position des occurrences de la chaîne passée en paramètre dans l'ensemble de K-mers indexé, et de retrouver la séquence correspondant au K-mer ayant l'identifiant passé en paramètre. Ligne 2 : initialisation d'un ensemble vide d'arcs. Lignes 4-5 : si le graphe est traversé en avant, les positions des occurrences du suffixe de longueur $k-1$ de s sont recherchées dans l'ensemble des K-mers. Lignes 6-7 : si le graphe est traversé en arrière, les positions des occurrences du préfixe de longueur $k-1$ de s sont recherchées dans l'ensemble des K-mers. Lignes 9-15 : Traitement de l'ensemble des positions des occurrences calculé à l'étape précédente. Le traitement s'arrête lorsque toutes les occurrences ont été traitées, ou que 4 arcs ont été trouvés, un sommet ne pouvant avoir plus de 4 arcs entrants ou sortants au sein d'un graphe de de Bruijn. Lignes 11-12 : si le graphe est traversé en avant, et que la position pos ne représente pas le suffixe de longueur $k-1$ du K-mer d'identifiant id, l'occurrence est étendue à droite, et un arc sortant est ajouté vers le k-mer démarrant en position pos du K-mer d'identifiant id. Lignes 13-14 : si le graphe est traversé en arrière, et que la position pos ne représente pas le préfixe de longueur $k-1$ du K-mer d'identifiant id, l'occurrence est étendue à gauche, et un arc entrant est ajouté à partir du k-mer démarrant en position $pos-1$ du K-mer d'identifiant id.

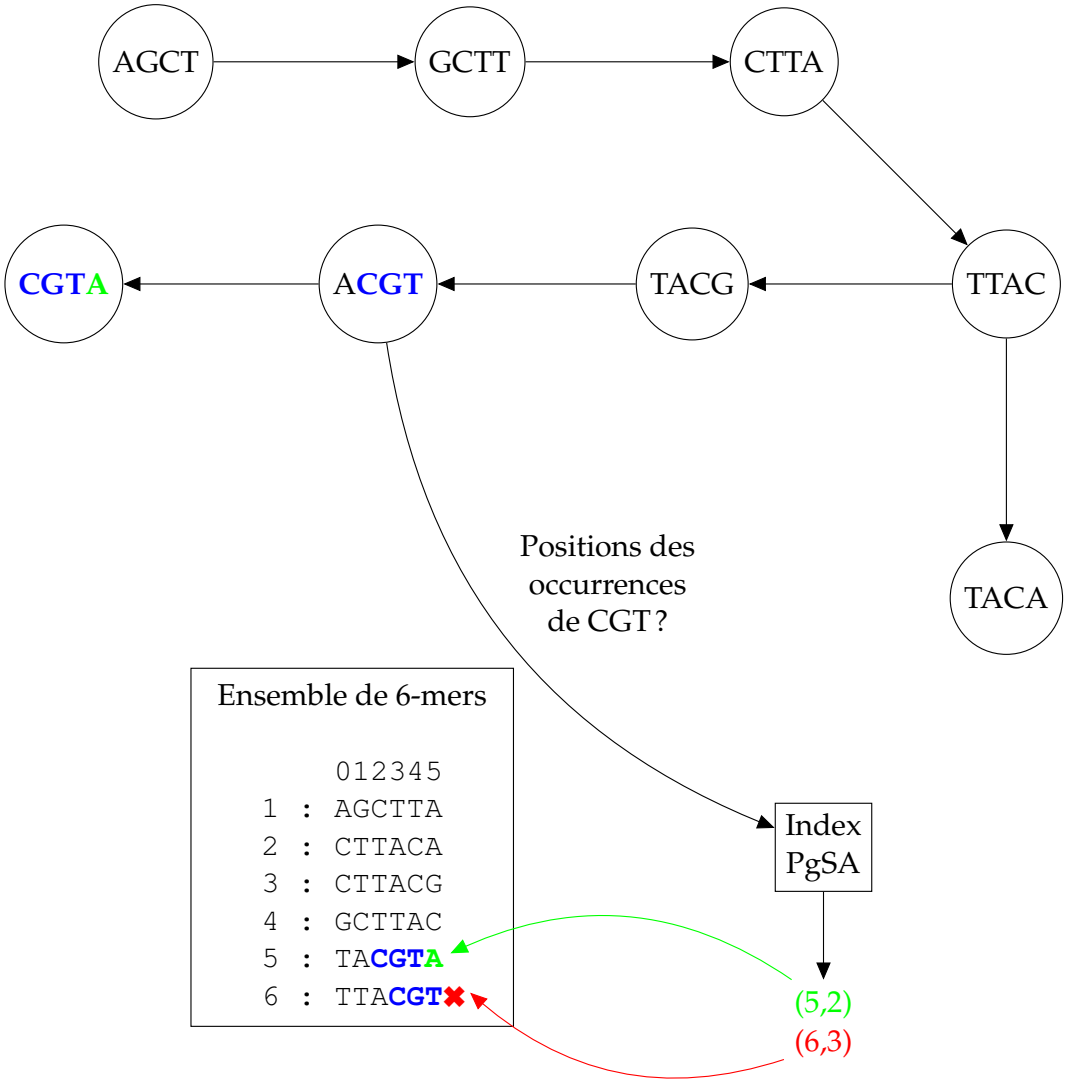

FIGURE 5.2 – **Traversée d'un graphe de de Bruijn d'ordre $k = 4$ à partir d'un graphe de de Bruijn d'ordre maximal $K = 6$ représenté par PgSA.** Le graphe est ici construit à partir de l'ensemble de *reads* $S = \{$AGCTTACA, CTTACGTA$\}$. Les 6-mers de ces *reads* sont donc extraits, et indexés avec PgSA, comme illustré dans la partie basse de la figure. Le graphe de de Bruijn d'ordre 4 de l'ensemble de *reads* S est représenté dans la partie haute de la figure. Pour trouver les arcs sortant du sommet ACGT au sein de ce graphe, les positions des occurrences de CGT, le suffixe de longueur 3 de ce sommet, sont recherchées à l'aide de PgSA. Un ensemble composé de deux paires d'entiers est ici retourné. La première paire, (5,2), correspond à une occurrence dans le $5^{\text{ème}}$ 6-mer de l'ensemble, en position 2. Cette occurrence peut être étendue à droite, afin de définir le 4-mer CGTA, et permet ainsi de retrouver l'arc sortant du sommet ACGT, et menant vers le sommet CGTA. La seconde paire, (6,3), correspond à une occurrence dans le $6^{\text{ème}}$ 6-mer de l'ensemble, en position 3. Cette occurrence ne peut pas être étendue à droite, puisque correspondant au suffixe de longueur 3 de ce 6-mer. Aucun arc sortant additionnel ne peut donc être défini.

5.3 Chaîne de traitement

5.3.1 Vue d'ensemble

Comme mentionné Section 5.1.3, HG-CoLoR combine différentes approches de l'état de l'art, à savoir l'alignement de *reads* courts sur les *reads* longs, et l'utilisation d'un graphe de de Bruijn, ayant la particularité d'être d'ordre variable. De plus, avant les étapes d'alignement et de construction du graphe, les *reads* courts sont également corrigés, afin d'optimiser la qualité de la correction des *reads* longs. HG-CoLoR repose ainsi sur une stratégie de type *seed and extend*, où les graines sont tout d'abord mises en évidence en alignant les *reads* courts sur les *reads* longs. Une graine est alors définie comme un 5-uple $(id, pos, long, score, seq)$, où :

- *id* représente l'identifiant du *read* long auquel la graine est associée.
- *pos* représente la position de début d'alignement de la graine sur le *read* long.
- *long* représente la longueur de l'alignement.
- *score* représente le score de l'alignement.
- *seq* représente la séquence de la graine s'alignant sur le *read* long.

Une fois les graines calculées, elles sont alors reliées en étendant leurs séquences, à l'aide du graphe de de Bruijn d'ordre variable. Comme indiqué Section 5.2.3, ce graphe est construit à partir des *reads* courts, en choisissant un ordre maximal K, et en indexant leurs K-mers avec PgSA, puis est traversé en interrogeant l'index. Pour chaque *read* long, le graphe est ainsi utilisé afin de relier les graines lui étant associées, celles-ci étant utilisées comme points d'ancrage sur le graphe. Comme dans les autres méthodes de correction basées sur des graphes de de Bruijn, le chemin suivi pour relier deux graines dicte alors une correction pour la région du *read* long non couverte, située entre ces deux graines. Enfin, une fois toutes les paires de graines reliées, les extrémités de la séquence obtenue sont étendues en traversant à nouveau le graphe, afin d'atteindre les extrémités du *read* long initial, et le *read* long corrigé est ainsi produit. Les différentes étapes constituant le pipeline de HG-CoLoR sont illustrées Figure 5.3, et détaillées ci-dessous.

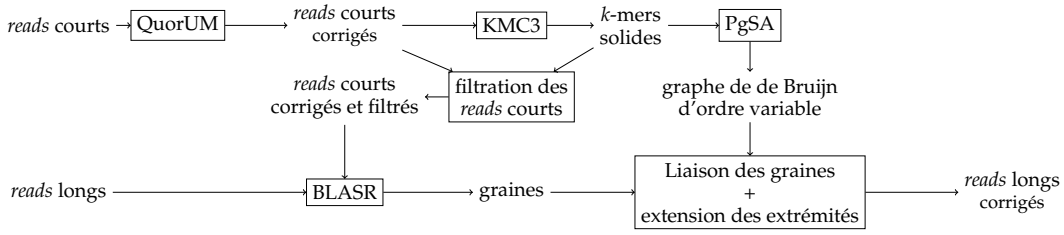

FIGURE 5.3 – **Pipeline de HG-CoLoR.** Tout d'abord, les *reads* courts sont corrigés avec QuorUM, afin de réduire l'impact des erreurs de séquençage qu'ils contiennent. Un ordre maximal K est ensuite choisi pour le graphe, et les K-mers des *reads* courts corrigés sont extraits à l'aide de KMC3. Afin de réduire davantage leur taux d'erreurs, une étape de filtration est appliquée aux *reads* courts corrigés, afin d'éliminer ceux contenant des K-mers faibles. De la même façon, pour réduire l'impact des erreurs sur le graphe, seul les K-mers solides des *reads* courts corrigés sont utilisés pour sa construction, et sont donc indexés avec PgSA. Les *reads* courts corrigés et filtrés précédemment sont ensuite alignés sur les *reads* longs, à l'aide de BLASR, afin de mettre en évidence les graines. Chaque *read* long est alors traité indépendamment. Pour chaque *read* long, le graphe est traversé, en utilisant les graines lui étant associées comme points d'ancrage, afin de relier celles-ci, et de corriger les régions non couvertes du *read*. Une fois toutes les graines reliées, les extrémités de la séquence obtenue sont étendues, en traversant à nouveau le graphe, afin d'atteindre les extrémités du *read* long initial. Le *read* long corrigé est alors reporté par HG-CoLoR.

5.3.2 Correction des *reads* courts

Bien que les *reads* courts soient déjà précis avant toute étape de correction, ils disposent toute de même d'un taux d'erreurs de l'ordre de 1%, pouvant introduire de potentiels biais lors de la correction, aussi bien lors de l'étape d'alignement que lors de la construction du graphe. En particulier, puisque HG-CoLoR vise à construire un graphe de de Bruijn avec un ordre maximal important, les erreurs contenues dans les *reads* courts doivent être corrigées, afin d'éviter l'introduction de chemins chimériques dans le graphe, pouvant impacter la qualité de la correction. En effet, en utilisant des *reads* courts, un graphe de de Bruijn d'ordre maximal $K = 100$ peut raisonnablement être construit. Cependant, le taux d'erreurs affiché par les *reads* courts impliquerait alors, en moyenne, la présence d'une erreur dans chaque K-mer. Bien que le fait de sélectionner uniquement les K-mers solides lors de la construction du graphe permette de réduire ce biais, une réelle étape de correction permet tout de même de limiter davantage l'impact des erreurs contenues dans les *reads* courts.

Pour cela, les *reads* courts sont donc corrigés à l'aide de QuorUM [82]. QuorUM réalise des corrections locales au sein des *reads* courts, en combinant l'analyse de la fréquence des leurs k-mers, avec un k de petite taille ($k = 24$ par défaut), et l'analyse des scores de qualité Phred de chaque base. En plus de ces corrections locales, visant à transformer les k-mers faibles en leurs k-mers solides les plus probables, les *reads* courts sont également *trimmés*, afin de réduire les biais liés aux erreurs présentes en majorité aux extrémités 3' des *reads* Illumina. QuorUM a été choisi pour son excellent compromis entre qualité de la correction et temps d'exécution, par rapport aux autres méthodes de correction de *reads* courts. Une comparaison entre QuorUM, Musket, Coral et HybridShrec et présentée Table 5.1. Ces résultats montrent la nette supériorité de QuorUM en termes de qualité de correction, l'identité moyenne des *reads* courts corrigés atteignant plus de 99,9% sur l'ensemble des jeux de données. Coral et HybridShrec, quant à eux, se sont montrés extrêmement coûteux, aussi bien en temps qu'en mémoire. En particulier, ces deux outils n'ont ainsi pas été évalués

Jeu de données	Correcteur	Nombre de bases (Mbp)	Identité (%)	Temps	Mémoire (Mo)
A. baylyi	Coral	224	99,91	18 min	7 565
	HybridShrec	215	99,81	41 min	3 942
	Musket	224	99,65	19 sec	75
	QuorUM	221	99,99	1 min	1 217
E. coli	Coral	232	99,74	18 min	8 214
	HybridShrec	230	99,64	41 min	3 985
	Musket	232	99,12	15 sec	77
	QuorUM	225	99,99	1 min	1 220
S. cerevisiae	Coral	625	99,75	38 min	11 386
	HybridShrec	607	99,10	2 h 06 min	6 392
	Musket	625	99,39	41 sec	72
	QuorUM	607	99,91	2 min	1 217
C. elegans	Coral	-	-	$>$ 12 h	-
	HybridShrec	-	-	$>$ 12 h	-
	Musket	5 014	98,34	5 min	79
	QuorUM	4 671	99,95	23 min	13 361

TABLE 5.1 – **Comparaison des performances de QuorUM, Musket, Coral et HybridShrec**. Les outils ont été évalués sur les *reads* courts des jeux de données de la Table 5.3. L'ensemble des outils de correction a été lancé sur 16 threads. L'identité moyenne des *reads* corrigés a été obtenue par alignement sur le génome de référence à l'aide de BWA-MEM. Coral et HybridShrec n'ont pas été évalués sur le jeu de données *C. elegans*, puisque nécessitant plus de 12 heures pour s'exécuter.

sur le jeu de données *C. elegans*, puisque nécessitant plus de 12 heures pour s'exécuter. Musket, de son côté, s'est montré extrêmement efficace en termes de temps d'exécution et de consommation de mémoire. Cependant, la qualité de la correction s'est révélée extrêment mauvaise par rapport aux autres méthodes, les *reads* courts corrigés atteignant une identité aussi faible que 98,34% sur le jeu de données *C. elegans*.

5.3.3 Construction du graphe

Une fois les *reads* courts corrigés, un ordre maximal K est choisi ($K = 100$ en pratique), et les K-mers de ces *reads* sont alors extraits à l'aide de KMC3 [54]. KMC3 a été choisi pour ses excellentes performances en termes de temps d'exécution et de consommation de mémoire, par rapport aux autres outils de l'état de l'art. Un comparatif entre KMC3, DSK [95] et Jellyfish [81] est dressé Table 5.2. Bien que KMC3 consomme davantage de mémoire que DSK sur le jeu de données *C. elegans*, il affiche cependant un net avantage sur ses deux concurrents en termes de temps d'exécution.

Afin de réduire davantage le taux d'erreurs des *reads* courts, et ainsi encore réduire l'impact de ces erreurs sur la correction des *reads* longs, les *reads* courts corrigés contenant des K-mers faibles sont filtrés. Ces *reads* courts ne sont donc pas utilisés dans les étapes suivantes, en particulier, dans l'étape d'alignement sur les *reads* longs permettant de mettre en évidence les graines, afin d'éviter l'utilisation d'ancres erronées lors des étapes d'extension suivantes. De la même façon, seuls les K-mers solides des *reads* courts corrigés sont utilisés pour représenter le graphe, et sont donc indexés avec PgSA, afin de réduire le nombre de chemins chimériques au sein du graphe.

5.3.4 Mise en évidence des graines

Afin de mettre en évidence l'ensemble de graines utilisé pour la correction de chaque *read* long, les *reads* courts, corrigés et filtrés par les deux étapes précédentes,

Outil	Performances	*A. baylyi*	*E. coli*	*S. cerevisiae*	*C. elegans*
DSK	Temps	23 sec	32 sec	1 min 23 sec	14 min 28 sec
	Mémoire (Mo)	2 354	2 693	3 451	5 280
Jellyfish	Temps	20 sec	30 sec	53 sec	56 min 23 sec
	Mémoire (Mo)	12 459	12 462	12 462	32 041
KMC3	Temps	15 sec	19 sec	37 sec	6 min 59 sec
	Mémoire (Mo)	793	1 132	2 123	11 249

TABLE 5.2 – **Comparaison des performances de DSK, Jellyfish et KMC3.** Les expériences ont été réalisées sur les *reads* courts des jeux de données de la Table 5.3. L'ensemble des outils de correction a été lancé sur 16 threads. Les extractions ont été réalisées avec $k = 100$.

sont alignés sur les *reads* longs à l'aide de BLASR. Après cette étape d'alignement, chaque *read* long est traité indépendamment, et deux phases d'analyse sont alors appliquées aux graines associées à chacun d'eux. Pour cela, les graines associées à chaque *read* long sont d'abord triées, dans l'ordre croissant de leur composante *pos*, décrivant la position de début d'alignement sur le *read* long. Cette étape est réalisée à l'aide de la fonction de tri du langage C++, de complexité asymptotique $\mathcal{O}(N\,log\,N)$.

5.3.4.1 Combinaison des graines ayant des positions d'alignement chevauchantes

Tout d'abord, les graines sont traitées par paires, dans l'ordre croissant de leurs composantes *pos*. Ainsi, si les positions et les longueurs d'alignement d'une paire de graines consécutives donnée indiquent que ces deux graines se chevauchent, sur une longueur suffisante L ($L = K - 1$ en pratique, afin de ne considérer que des chevauchements significatifs), et que leurs séquences chevauchantes sont identiques, les deux graines sont alors combinées en une graine unique. Au contraire, si les séquences chevauchantes de ces deux graines ne sont pas identiques, ou que les positions et les longueurs des alignements indiquent un chevauchement de longueur insuffisante, seule la graine possédant le plus haut score est conservée. L'algorithme décrivant cette première étape de combinaison des graines est donné Figure 5.4.

5.3.4.2 Combinaison des graines partageant des chevauchements préfixe / suffixe

Ensuite, une fois cette première étape de traitement terminée, les graines résultantes sont de nouveau traitées par paires, dans l'ordre croissant de leurs composantes *pos*. Les chevauchements préfixe / suffixe entre les graines ayant des positions de début et de fin d'alignement proches (< 10 bases en pratique) sont alors calculés. De la même façon que lors de l'étape précédente, si le suffixe d'une graine donnée chevauche parfaitement le préfixe de la graine suivante, sur une longueur suffisante L (toujours avec $L = K - 1$ en pratique), les deux graines sont alors combinées en une graine unique. Cette étape d'analyse permet de prendre en compte les petites erreurs d'insertions contenues dans les *reads* longs n'ayant pas été détectées lors de l'étape d'alignement, et pouvant compliquer l'étape suivante d'extension des graines. L'algorithme décrivant cette seconde étape de combinaison des graines est donné Figure 5.5.

```
COMBINAISON DES GRAINES SELON LEURS POSITIONS D'ALIGNEMENT(S, longMin)
▷ Entrée : S tableau de 5-uple (chaîne, entier, entier, entier, entier), longMin entier
▷ Sortie : L'ensemble de graines après combinaison des graines chevauchantes
▷ Auxiliaire : i entier, j entier, s1 chaîne, s2 chaîne, longueurSuff entier,
▷                sequenceSuff chaîne, scoreSuff entier
1  Début
2    i ← 0
3    j ← 1
4    tantque i < LONGUEUR(S) − 1 et j < LONGUEUR(S) faire
5      si S[j].pos ≤ S[i].pos + S[i].long alors
6        s1 ← S[i].seq[S[j].pos − S[i].pos..S[i].long − 1]
7        s2 ← S[j].seq[0..long(s1) − 1]
8        si S[j].pos + S[j].long > S[i].pos + S[i].long et LONGUEUR(s1) ≥ longMin
           et s1 = s2 alors
9          longueurSuff ← S[j].pos + S[j].long − S[i].pos − S[i].long
10         sequenceSuff ← S[j].seq[S[j].long − suffLen..S[j].long − 1]
11         scoreSuff ← (S[j].score/S[j].long) × suffLen
12         S[i].seq ← S[i].seq + sequenceSuff
13         S[i].long ← S[i].long + longueurSuff
14         S[i].score ← S[i].score + scoreSuff
15         SUPPRIMER(S[j])
16       sinon si S[i].score < S[j].score alors
17         SUPPRIMER(S[j])
18       sinon
19         SUPPRIMER(S[i])
20     sinon
21       i ← j
22       j ← j + 1
23   Fin
```

FIGURE 5.4 – **Algorithme de combinaison des graines ayant des positions d'alignement chevauchantes.** Lignes 2-3 : début du traitement, en sélectionnant les deux premières graines. Lignes 5-19 : les positions d'alignement des deux graines courantes se chevauchent, les deux graines peuvent alors potentiellement être combinées. Lignes 6-7 : Récupération des séquences chevauchantes des deux graines. Lignes 8-15 : si les positions d'alignement de la $j^{\text{ème}}$ graine ne sont pas incluses dans celles de la $i^{\text{ème}}$ graine, si ces deux graines se chevauchent sur une longueur suffisante, et si les séquences chevauchantes de ces deux graines sont identiques, les deux graines sont combinées. Lignes 9-11 : récupération de la longueur, de la séquence, et du score du suffixe de la $j^{\text{ème}}$ graine non inclus dans le chevauchement. Le score du suffixe est défini comme le score moyen d'une base (*i.e.* le score total divisé par la longueur de la séquence de la graine) multiplié par la longueur de ce suffixe. Lignes 12-14 : combinaison des graines. Le suffixe non chevauchant de la $j^{\text{ème}}$ graine est concaténé à la fin de la séquence de la $i^{\text{ème}}$ graine, et la longueur d'alignement et le score de cette graine sont mis à jour en fonction de cette concaténation. Ligne 15 : la $i^{\text{ème}}$ et la $j^{\text{ème}}$ ont été combinées, la $j^{\text{ème}}$ graine est donc supprimée. Lignes 16-19 : les graines ne peuvent pas être combinées, la graine ayant le score le plus faible est alors supprimée. Lignes 20-22 : les positions d'alignement des graines ne se chevauchent pas, la paire de graines suivante est alors considérée.

```
COMBINAISON DES GRAINES SELON LEURS CHEVAUCHEMENTS(S, distMax, longMin)
▷ Entrée : S tableau de 5-uple (chaîne, entier, entier, entier, entier), distMax entier,
▷          longMin entier
▷ Sortie : L'ensemble de graines après combinaison des graines chevauchantes
▷ Auxiliaire : i integer, j integer, longueurChev entier, longueurSuff entier,
▷              sequenceSuff chaîne, scoreSuff entier
1  Début
2    i ← 0
3    j ← 0
4    tantque i < LONGUEUR(S) − 1 et j < LONGUEUR(S) faire
5      si S[j].pos − S[i].pos − S[i].len ≤ distMax alors
6        longueurChev ← LONGUEURDUCHEVAUCHEMENT(S[i].seq, S[j].seq))
7        si longueurChev ≥ longMin alors
8          longueurSuff ← S[j].len − longueurChev
9          sequenceSuff ← S[j].seq[overlap..S[j].len − 1]
10         scoreSuff ← (S[j].score/S[j].len) × longueurSuff
11         S[i].seq ← S[i].seq + sequenceSuff
12         S[i].long ← S[i].len + longueurSuf
13         S[i].score ← S[i].score + scoreSuff
14         SUPPRIMER(S[j])
15       sinon
16         i ← j
17         j ← j + 1
18     sinon
19       i ← j
20       j ← j + 1
21 Fin
```

FIGURE 5.5 – **Algorithme de combinaison des graines partageant des chevauchements préfixe / suffixe.** Lignes 2-3 : début du traitement, en sélectionnant les deux premières graines. Lignes 5-17 : les deux graines courantes ont des positions d'alignement proches, elles peuvent donc potentiellement être combinées. Ligne 6 : calcul de la longueur du chevauchement préfixe / suffixe entre les deux graines. Lignes 7-14 : si le chevauchement préfixe / suffixe est suffisamment long, les deux graines sont combinées. Lignes 8-10 : récupération de la longueur, de la séquence, et du score du suffixe de la $j^{\text{ème}}$ graine non inclus dans le chevauchement. Ici encore, le score du suffixe est défini comme le score moyen d'une base multiplié par la longueur de ce suffixe. Lignes 11-13 : combinaison des graines. Comme lors de l'étape précédente, le suffixe non chevauchant de la $j^{\text{ème}}$ graine est concaténé à la fin de la séquence de la $i^{\text{ème}}$ graine, et la longueur d'alignement et le score de cette graine sont mis à jour. Ligne 14 : la $i^{\text{ème}}$ et la $j^{\text{ème}}$ ont été combinées, la $j^{\text{ème}}$ graine est donc supprimée. Lignes 15-17 : le chevauchement préfixe / suffixe entre les graines est trop court, et les graines ne peuvent donc pas être combinées. La paire de graines suivante est alors considérée. Lignes 18-20 : les positions d'alignement des deux graines courantes sont trop éloignées, et les graines ne peuvent pas être combinées

5.3.5 Extension et liaison des graines

5.3.5.1 Principe

Une fois les graines mises en évidence et les deux étapes de combinaison réalisées, pour un *read* long donné, HG-CoLoR étend alors ces graines afin de les relier, en les considérant par paires, et en traversant le graphe. Pour une paire de graines donnée, la graine ayant la position d'alignement la plus à gauche est appelée la *source*, et la graine ayant la position d'alignement la plus à droite est appelée la *cible*.

Pour relier une paire de graines, le suffixe de la source et le préfixe de la cible sont utilisés comme points d'ancrage sur le graphe. Celui-ci est alors traversé, afin de trouver un chemin entre ces deux ancres. Quand un tel chemin est trouvé, la séquence dictée par la suite de sommets traversés est alors utilisée comme correction pour la région non couverte du *read* long située entre les deux graines. La recherche d'un chemin entre deux graines est d'abord réalisée de la source vers la cible. Si cette recherche ne permet pas de mettre un évidence un chemin, une nouvelle recherche est alors effectuée, cette fois de la cible vers la source. Comme dans les autres méthodes basées sur des graphes de de Bruijn, notamment LoRDEC, LoRMA et FMLRC, la recherche est ainsi effectuée dans les deux directions, car en fonction du point de départ, différents sous-graphes peuvent être explorés, et ainsi mener à des traversées différentes. Le principe d'extension et de liaison des graines adopté par HG-CoLoR est illustré Figure 5.6.

5.3.5.2 Modification de l'ordre du graphe

HG-CoLoR traverse toujours le graphe de de Bruijn d'ordre variable en commençant par l'ordre le plus élevé. L'ordre du graphe est réduit à un sommet donné uniquement si ce sommet ne possède pas d'arc sortant pour l'ordre courant, ou si tous ses arcs sortants pour l'ordre courant ont déjà été explorés, et n'ont pas permis de mettre en évidence un chemin entre la cible et la source. Lorsque l'ordre du graphe est réduit, la taille des k-mers de la source et de la cible utilisés comme ancres est également réduite, afin de pouvoir de nouveau ancrer les deux graines sur le graphe. Un ordre minimal est également défini (40 en pratique), afin que HG-CoLoR ne traverse pas de graphes de de Bruijn d'ordres trop faibles, contenant trop d'embranchements, et représentant des chevauchements trop courts et trop peu informatifs.

Quand la traversée du graphe, pour un ordre k, amène à un embranchement, HG-CoLoR réalise une sélection gloutonne. Ainsi, l'arc menant vers le sommet représentant le k-mer ayant le plus grand nombre d'occurrences est toujours exploré en premier. Cette sélection gloutonne permet d'éviter de traverser trop de sommets représentant des k-mers ayant de faibles fréquences d'apparition, et qui, malgré les deux étapes de correction et de filtration, sont tout de même susceptibles de contenir des erreurs. De plus, à la suite de la traversée d'un arc provenant d'un graphe de de Bruijn d'ordre $k < K$, dû à un manque d'arcs permettant de relier deux graines au sein des graphes d'ordres plus élevés, l'ordre du graphe est toujours réinitialisé à l'ordre maximal K avant de poursuivre la traversée. Cette réinitialisation permet d'éviter de traverser de larges régions au sein des graphes de de Bruijn d'ordre inférieur à K, représentant des chevauchements plus courts, et étant donc susceptibles de contenir de plus nombreux embranchements, et de compliquer la recherche de chemin.

Quand un chemin entre deux graines est trouvé, il est alors considéré comme optimal, grâce au processus de sélection gloutonne, et au fait que l'ordre du graphe ne soit réduit que localement, permettant ainsi d'éviter au maximum la traversée d'arcs

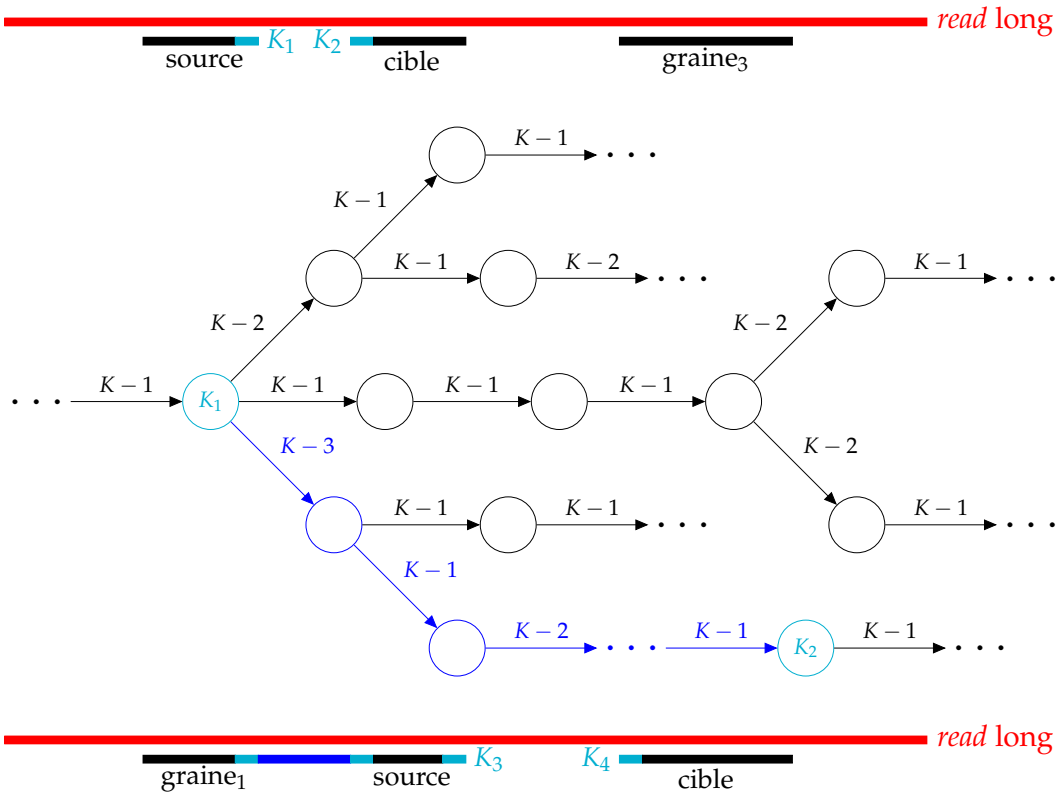

FIGURE 5.6 – **Processus d'extension et de liaison des graines.** Partie haute : une paire de graines est sélectionnée, une de ces graines définissant la source, et l'autre définissant la cible de l'extension. Le suffixe de la source (K_1), et le préfixe de la cible (K_2) sont alors utilisés comme points d'ancrage sur le graphe. Partie du milieu : le graphe est traversé, afin de mettre en évidence un chemin entre K_1 et K_2. Ce chemin est ici représenté en bleu foncé. Les poids des arcs du graphe correspondent aux différents ordres pouvant être sélectionnés durant la traversée. Partie basse : le chemin du graphe permettant de relier K_1 et K_2 est utilisé comme correction pour la partie non couverte du *read* long, située entre les deux ancres. La paire de graines suivante est alors considérée, définissant ainsi une nouvelle source et une nouvelle cible. Comme précédemment, les K-mers situés aux extrémités de ces graines (K_3 et K_4) sont utilisés comme point d'ancrage sur le graphe. Celui-ci est alors traversé afin de rechercher un chemin entre ces deux nouvelles ancres.

représentant des chevauchements courts. La séquence dictée par les sommets traversés par ce chemin est alors choisie comme correction pour la région non couverte du *read* long, située entre les deux graines. Le chemin optimal est volontairement choisi de cette façon, plutôt qu'à la façon de LoRDEC ou FMLRC, explorant plusieurs chemins et sélectionnant comme correction celui s'alignant le mieux avec la région non couverte du *read* long, afin de réduire le nombre de traversées du graphe, et donc le temps d'exécution.

5.3.5.3 Prise en compte des graines contenant des erreurs

Malgré les étapes de correction et de filtration des *reads* courts, une faible proportion d'erreurs peut encore être présente au sein des *reads* courts corrigés, et donc au sein des graines. Pour limiter l'impact de ces erreurs, un plafond de substitutions m ($m = 3$ en pratique) est également défini lors de la recherche d'un chemin permettant de relier une paire de graines donnée. De cette façon, la source et la cible sont considérées comme pouvant être reliées si un chemin partant du K-mer ancre de la source (resp. de la cible) permet d'atteindre un K-mer affichant moins de m substitutions par rapport au K-mer ancre de la cible (resp. de la source). Ce plafond permet ainsi de surmonter les difficultés liées aux quelques erreurs de substitutions pouvant encore être présentes au sein des graines, et pouvant alors mener à des recherches de chemins inexistants dans le graphe. Cependant, un tel procédé permet de traiter uniquement les cas où seule une des deux graines comporte des erreurs, et n'est pas applicable dans les cas où des erreurs sont présentes dans les deux graines. Une autre stratégie permettant de prendre en compte ces cas problématiques est présentée Section 5.3.5.4.

5.3.5.4 Saut de graines

HG-CoLoR peut être amené à rechercher un chemin entre deux graines apparaissant dans des régions non connectées du graphe. Cette situation peut se produire lorsque les deux graines d'une paire donnée contiennent des erreurs, la stratégie proposée Section 5.3.5.3 ne permettant pas de traiter ces cas, mais également lorsqu'un *read* court provenant d'une région différente du génome a été aligné par erreur sur un *read* long donné.

Ainsi, afin d'éviter des explorations intensives et coûteuses du graphe, notamment dans ces cas, une limite ℓ ($\ell = 1250$ en pratique) sur le nombre maximal d'embranchements du graphe à explorer est définie. Si cette limite est atteinte, et qu'aucun chemin n'a pu être trouvé pour relier la source et la cible, l'étape d'extension de ces deux graines est abandonnée, et HG-CoLoR essaie alors de sauter la cible n'ayant pas pu être atteinte. En d'autres termes, la source reste la même, la cible n'ayant pas pu être atteinte est ignorée, et la cible est alors redéfinie comme la graine suivante. Une nouvelle étape d'extension est alors appliquée à ces deux graines, et un chemin est ainsi à nouveau recherché au sein du graphe, afin de relier la source à cette nouvelle cible. Ce processus est illustré Figure 5.7.

Ce processus de saut de graines peut cependant mener à un nombre important de tentatives d'extension ratées (*i.e.* une impossibilité de relier deux graines en respectant le nombre maximal l d'embranchements à explorer). Ainsi, un nombre maximum de graines pouvant être sautées s ($s = 5$ en pratique) est également défini. Quand ce nombre est atteint, et qu'aucune des étapes d'extension n'a permis de mettre en évidence un chemin entre deux graines, HG-CoLoR reporte la région non couverte du *read* long original, entre la source et la première cible ayant été sautée,

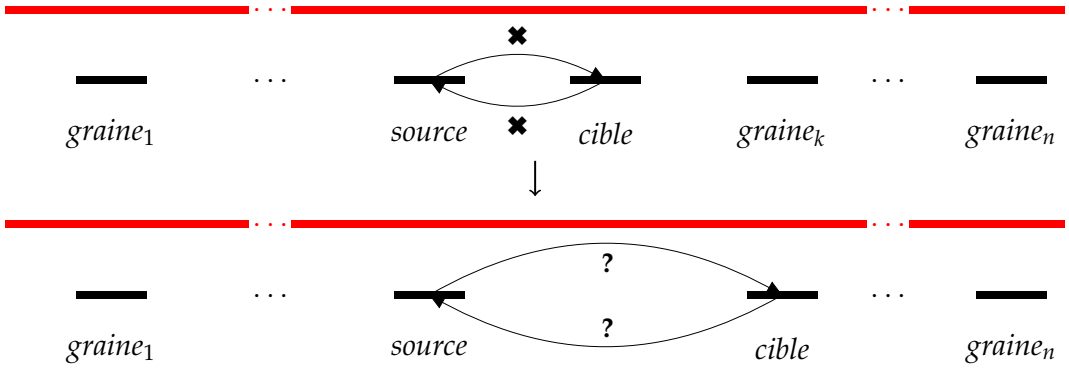

FIGURE 5.7 – **Processus de saut de graines.** Les segments rouges représentent le *read* long, et les segments noirs représentent les graines. Partie haute : aucun chemin permettant de relier la graine *source* et la graine *cible* n'a été trouvé. La graine *cible* va donc être sautée. Partie basse : la graine *cible* est redéfinie comme la $graine_k$, et la graine *source* reste inchangée. Une nouvelle étape d'extension est alors appliquée, en recherchant un chemin de la graine *source* vers la graine *cible*, et, si nécessaire, de la graine *cible* vers la graine *source*.

dans la séquence du *read* long corrigé. Le graphe est ensuite de nouveau traversé, afin de relier les graines et de corriger les régions restantes du *read* long, en recommençant les extensions à partir de la première graine ayant été sautée.

5.3.6 Extension des extrémités

Les graines associées à un *read* long ne couvrent pas toujours l'intégralité de la longueur de celui-ci. En particulier, les extrémités des *reads* longs sont rarement intégralement couvertes par des graines. Ainsi, pour conserver autant que possible la longueur des *reads* longs originaux, une fois que l'ensemble des graines d'un *read* long donné a été traité, HG-CoLoR continue à traverser le graphe afin d'étendre les extrémités du *read* long corrigé ainsi produit.

À la manière de LoRDEC, de LoRMA, et de FMLRC, l'extension des extrémités commence ainsi à partir du premier (resp. du dernier) K-mer du *read* long corrigé, représentant tous deux des K-mers issus des graines, et étant donc de haute qualité. Comme lors de l'étape précédente, la traversée du graphe commence alors à partir de l'ordre maximal, et l'ordre du graphe est réduit à un sommet donné uniquement ci celui-ci ne possède pas d'arc sortant pour l'ordre courant. Les extrémités du *read* long corrigé sont ainsi étendues jusqu'à ce que les extrémités du *read* long original soient atteintes, ou jusqu'à ce qu'une pointe ou un embranchement soient rencontrés au sein du graphe. En effet, lors de l'extension des extrémités du *read* long corrigé, HG-CoLoR ne dispose d'aucune information concernant le chemin à suivre au niveau d'un embranchement, et ne dispose pas non plus d'ancres permettant d'explorer plusieurs chemins, comme lors de l'étape précédente. Ainsi, la sélection gloutonne et l'exploration de multiples chemins est inutile, et l'extension du *read* long corrigé est alors stoppée lorsqu'un embranchement est rencontré. Dans ces cas, les extrémités du *read* long corrigé sont alors étendues à l'aide des bases du *read* long original, afin de conserver l'intégralité de la longueur de ce dernier.

De plus, l'extension est toujours arrêtée une fois les extrémités du *read* long original atteintes, même dans les cas où le graphe pourrait encore être traversé sans ambiguïté. En effet, HG-CoLoR a pour but de corriger les *reads* longs, et non de réaliser un assemblage, et continuer à étendre les extrémités du *read* long corrigé au

delà des extrémités du *read* long original s'assimilerait alors davantage à un processus d'assemblage qu'à un processus de correction.

5.3.7 Sortie

Reporter les bases des *reads* longs originaux au sein des *reads* longs corrigés quand aucun chemin ne peut être trouvé entre deux graines, et étendre les extrémités des *reads* longs corrigés avec les bases des *reads* longs originaux permet de conserver l'intégralité de l'information portée par la longueur des *reads*. Un tel procédé peut être intéressant pour les applications où la longueur des *reads* joue un rôle important, comme par exemple pour l'assemblage. Pour les autres applications, où la qualité des séquences est plus importantes que leur longueur, HG-CoLoR permet également de *trimmer* et de *splitter* les *reads* longs corrigés.

En effet, à la fin du processus de correction, chaque base d'un *read* long corrigé peut être considérée comme corrigée, si elle provient d'une graine ou d'une traversée du graphe, ou comme non corrigée si elle provient du *read* long original. HG-CoLoR reporte ainsi les bases corrigées en majuscules et les bases non corrigées en minuscules. Les *reads* longs corrigés peuvent ainsi être *trimmés* et / ou *splittés* après la correction, en retirant les bases non corrigées situées aux extrémités des *reads*, ou en ne conservant que l'ensemble des régions des *reads* ayant effectivement pu être corrigées. En pratique, HG-CoLoR produit, en sortie, les trois différentes versions de chaque *read* long corrigé.

5.4 Résultats

Nous comparons les résultats produits par HG-CoLoR aux résultats produits par divers outils de correction hybride de l'état de l'art partageant des similarités avec HG-CoLoR. LoRDEC et FMLRC sont ainsi inclus dans la comparaison, afin d'étudier l'impact du type de graphe de de Bruijn utilisé sur la correction. CoLoRMap et NaS sont également inclus, puisqu'utilisant tous deux des approches permettant de réduire les biais liés aux régions non couvertes des *reads* longs, CoLoRMap en faisant usage des informations portées par les *reads* pairés, et NaS en comparant directement les *reads* courts entre eux. Enfin, deux outils d'auto-correction, MECAT et Daccord, adoptant respectivement une approche de correction par alignement multiple et une approche de correction par graphe de de Bruijn sont également inclus.

Ces différents outils sont comparés sur divers jeux de données, d'espèces de taille et de complexité croissantes (*A. baylyi*, *E. coli*, *S. cerevisiae* et *C. elegans*), et composés aussi bien de *reads* simulés PacBio que de *reads* réels ONT. Les jeux de données simulées utilisés ici sont identiques à ceux utilisés au sein du Chapitre 4, et présentés Table 4.4. Ils ont donc été simulés à l'aide de SimLoRD, en leur affectant un taux d'erreurs de 18,6%, et une profondeur de séquençage de 20x. L'ensemble des jeux de données, réels et simulés, utilisés pour ces expériences est présenté Table 5.3.

Un *benchmark* plus complet, comparant l'intégralité des outils de correction hybride et d'auto-correction sur l'ensemble de ces jeux de données est également proposé dans le Chapitre 7, Section 7.2.

Tous les outils de correction dont les résultats sont présentés ici, aussi bien sur données simulées que sur données réelles, ont été lancés en utilisant leurs paramètres par défaut ou les paramètres recommandés dans leurs publications respectives, sur une machine disposant de 32 Go de RAM, et avec 16 threads. Le processus

Jeu de données	*A. baylyi*	*E. coli*	*S. cerevisiae*	*C. elegans*
Génome de référence				
Souche	ADP1	K-12 substr. MG1655	W303	Bristol N2
Séquence de référence	CR543861	NC_000913	scf7180000000{084-13}	GCA_000002985.3
Taille du génome (Mbp)	3,6	4,6	12,2	100
***Reads* simulés PacBio**				
Nombre de *reads*	8 765	11 306	30 132	244 277
Longueur moyenne (bp)	8 202	8 226	8 204	8 204
Nombre de bases (Mbp)	72	93	247	2 004
Couverture	20x	20x	20x	20x
Taux d'erreurs (%)	18,6	18,6	18,6	18,6
***Reads* réels ONT**				
Numéro d'accession	Genoscope[1]	Genoscope[2]	Genoscope[3]	ERR1802061[4]
Nombre de *reads*	89 011	22 270	205 923	363 500
Longueur moyenne (bp)	4 284	5 999	5 698	5 524
Nombre de bases (Mbp)	381	134	1 173	2 008
Couverture	106x	29x	95x	20x
Taux d'erreurs (%)	29,91	20,54	44,51	28,93
***Reads* Illumina**				
Numéro d'accesion	ERR788913[5]	Genoscope[6]	Genoscope[7]	ART
Nombre de *reads*	900 000	775 500	2 500 000	20 057 100
Longueur des *reads* (bp)	250	300	250	250
Nombre de bases (Mbp)	224	232	625	5 000
Couverture	50x	50x	50x	50x

TABLE 5.3 – **Description des jeux de données utilisés pour les comparaisons entre HG-CoLoR et les outils de correction de l'état de l'art.**

[1] http://www.genoscope.cns.fr/externe/nas/datasets/MinION/acineto/
[2] http://www.genoscope.cns.fr/externe/nas/datasets/MinION/ecoli/
[3] http://www.genoscope.cns.fr/externe/nas/datasets/MinION/yeast/
[4] http://www.genoscope.cns.fr/externe/nas/datasets/Illumina/acineto/
[6] http://www.genoscope.cns.fr/externe/nas/datasets/Illumina/ecoli/
[7] http://www.genoscope.cns.fr/externe/nas/datasets/Illumina/yeast/

de correction de HG-CoLoR, durant les étapes de liaison des graines et d'extension, peut en effet aisément être parallélisé, en traitant un *read* différent sur chaque thread.

5.4.1 Impact des paramètres

Nous étudions ici l'impact des principaux paramètres (ordre maximal et minimal du graphe, nombre maximal d'embranchements à explorer) sur les résultats fournis par HG-CoLoR. Des expériences comparatives ont été réalisées, sur le jeu de données réelles *S. cerevisiae* de la Table 5.3.

L'impact de l'ordre maximal du graphe est illustré Figure 5.8. Ces résultats soulignent qu'un ordre maximal fixé à $K = 100$ produit davantage de *reads splittés*, affichant donc une longueur moyenne légèrement plus courte, mais également une identité plus faible qu'un ordre maximal fixé à $K = 90$. Cependant, fixer l'ordre maximal à $K = 100$ permet au processus de correction de réduire de près d'une heure le temps d'exécution, and fournit ainsi un compromis satisfaisant. Par rapport à ces deux valeurs, choisir un ordre maximal plus élevé permet d'obtenir des *reads* corrigés affichant une plus haute identité, mais induit une diminution du nombre de *reads* corrigés, et donc du nombre de bases corrigées produites. À l'inverse, choisir un ordre maximal moins élevé permet de corriger davantage de *reads*, mais augmente également la proportion de *reads splittés* et le temps d'exécution.

L'impact de l'ordre minimal du graphe est illustré Figure 5.9. Ces résultats soulignent qu'excepté le taux de couverture du génome, et l'identité moyenne des *reads*

corrigés, l'ensemble des autres statistiques affichent un net pic lorsque l'ordre minimal du graphe est fixé à $k = 40$.

L'impact du nombre maximal d'embranchements à explorer est illustré Figure 5.10. Ces résultats soulignent que la longueur moyenne des *reads*, le nombre total de bases corrigées ainsi que le temps d'exécution, augmente proportionnellement avec le nombre maximal d'embranchemets. L'augmentation de ce nombre impacte cependant également l'identité moyenne des *reads* corrigés et la proportion de *reads splittés*, celles-ci diminuant en effet à mesure que le nombre maximal d'embranchements augmente. Un nombre maximal d'embranchements fixé à $\ell = 1250$ offre ainsi un bon compromis entre l'ensemble de ces métriques.

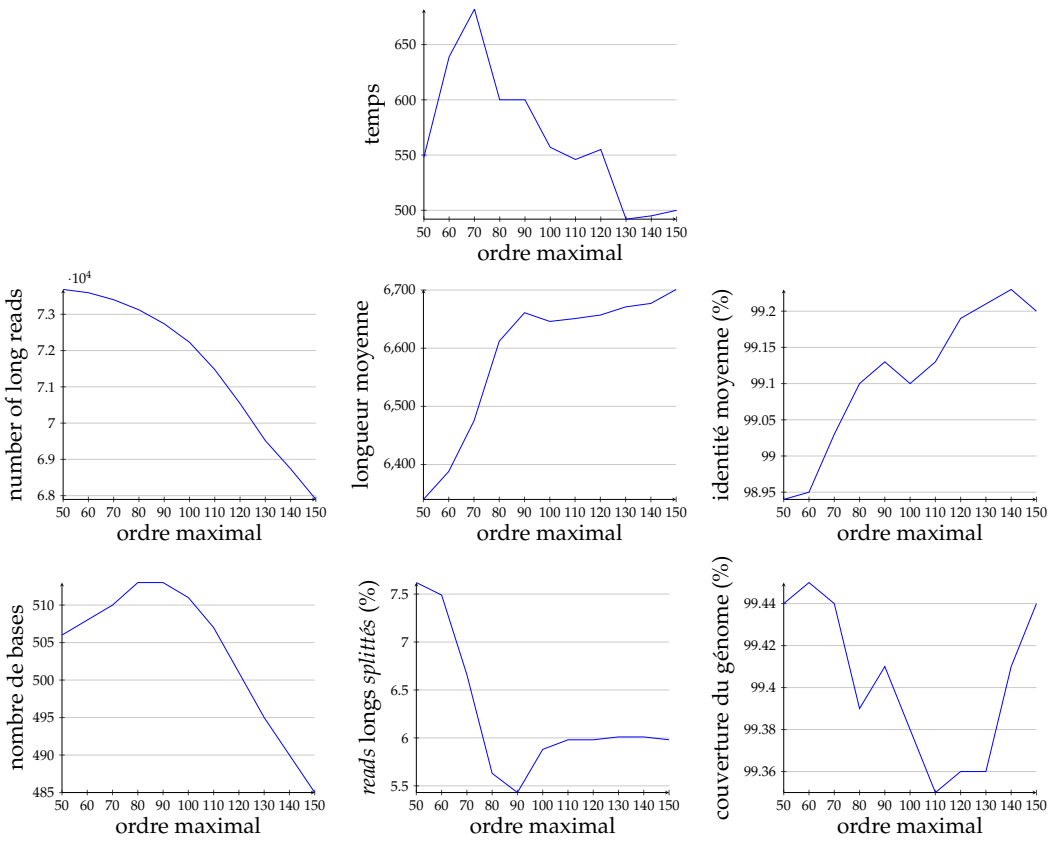

FIGURE 5.8 – **Impact de l'ordre maximal du graphe sur les résultats, en fixant les autres paramètres.** Pour obtenir des comparaisons équitables, l'ordre minimal du graphe a été fixé à la moitié de l'ordre maximal, pour chaque expérience. Les temps d'exécution sont reportés pour l'intégralité du processus de correction.

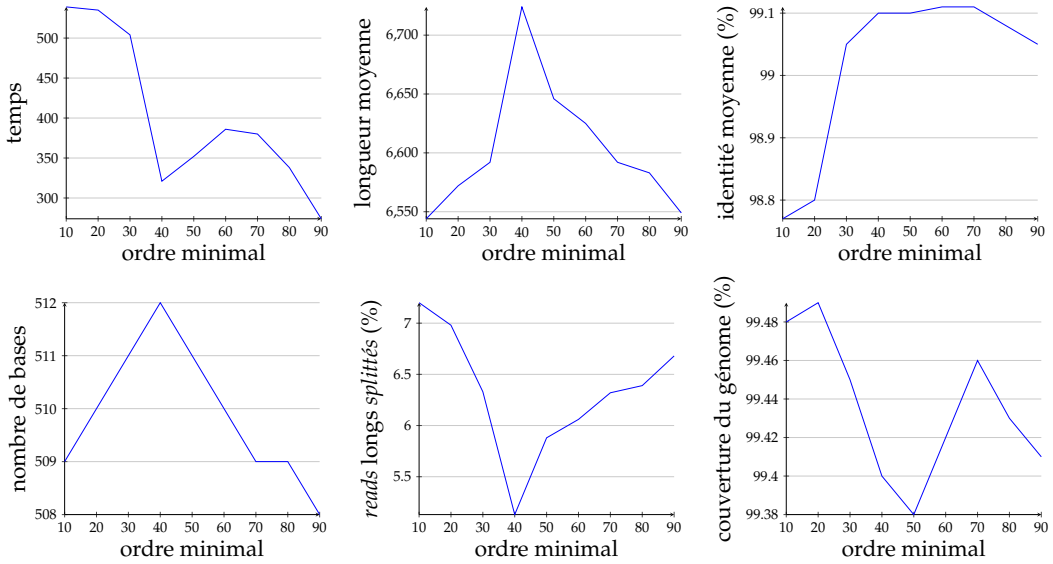

FIGURE 5.9 – **Impact de l'ordre minimal du graphe sur les résultats, en fixant les autres paramètres.** L'ordre maximal du graphe a été fixé à $K = 100$ pour l'ensemble des expériences. Les temps d'exécution ne sont reportés que pour les étapes de liaison des graines et d'extension des extrémités. L'impact du choix de l'ordre minimal sur le nombre de *reads* longs corrigés n'est pas illustré, car toutes les valeurs ont produit le même nombre de *reads*.

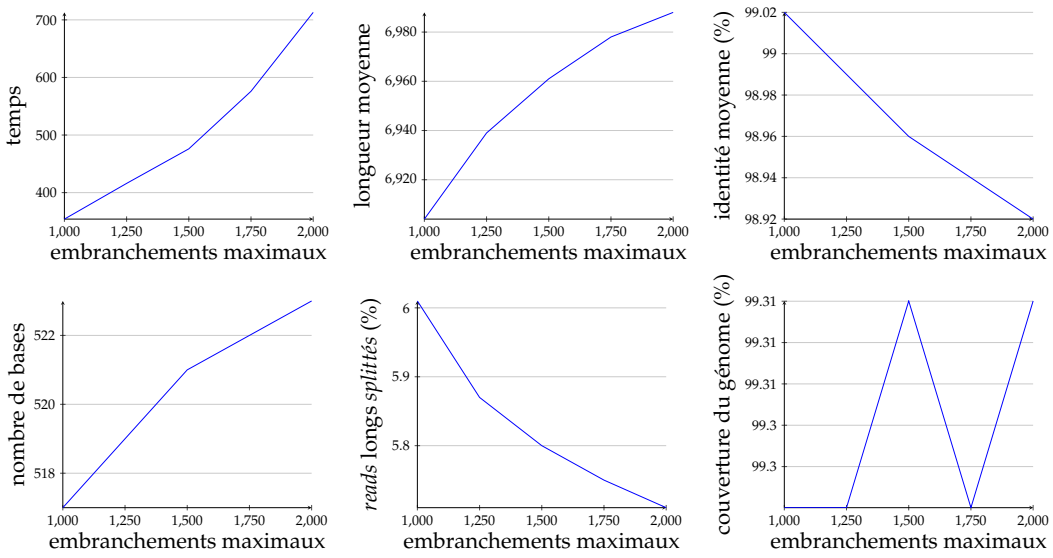

FIGURE 5.10 – **Impact du nombre maximal d'embranchements à explorer sur les résultats, en fixant les autres paramètres.** L'ordre maximal et l'ordre minimal du graphe ont été fixés, respectivement, à $K = 100$ et $k = 40$. Les temps d'exécution ne sont reportés que pour les étapes de liaison des graines et d'extension des extrémités. L'impact du choix du nombre maximal d'embranchements sur le nombre de *reads* longs corrigés n'est pas illustré, car toutes les valeurs ont produit le même nombre de *reads*.

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Daccord	Nombre de bases	64 669 977	83 773 362	222 050 951	-
	Taux d'erreurs (%)	0,4694	0,3965	0,5447	-
	Rappel (%)	99,8647	99,8817	99,8591	-
	Précision (%)	99,5371	99,6077	99,4630	-
	Reads trimmés ou *splittés*	328	119	991	-
	Reads étendus	0	0	0	-
	Temps	16 min	24 min	1 h 10 min	-
	Mémoire (Mo)	3 509	4 538	14 111	-
MECAT	Nombre de bases	46 854 371	58 979 203	162 057 920	870 965 775
	Taux d'erreurs (%)	0,5340	0,5243	0,6555	0,6540
	Rappel (%)	99,8289	99,8317	99,8015	99,8196
	Précision (%)	99,4817	99,4915	99,3636	99,3597
	Reads trimmés ou *splittés*	4 802	6 304	16 034	120 325
	Reads étendus	0	0	1	41
	Temps	22 sec	26 sec	1 min 20 sec	18 min 20 sec
	Mémoire (Mo)	1 202	1 322	2 207	10 340
CoLoRMap	Nombre de bases	64 353 820	81 060 058	210 966 804	571 054 288
	Taux d'erreurs (%)	0,1336	0,1946	0,2655	2,6255
	Rappel (%)	99,9923	99,9890	99,9805	99,8445
	Précision (%)	99,8707	99,8118	99,7413	97,4526
	Reads trimmés ou *splittés*	1 666	2 067	6 970	64 051
	Reads étendus	8 463	10 730	28 117	96 953
	Temps	57 min	1 h 25 min	4 h 42 min	125 h 44 min
	Mémoire (Mo)	6 198	9 659	13 544	32 188
NaS	Nombre de bases	58 963 977	78 034 042	207 085 378	-
	Taux d'erreurs (%)	2,1095	1,3035	2,7483	-
	Rappel (%)	99,9158	99,9468	99,9077	-
	Précision (%)	97,8969	98,7009	97,2615	-
	Reads trimmés ou *splittés*	339	274	2 187	-
	Reads étendus	6 688	8 070	22 658	-
	Temps	24 h 24 min	28 h 48 min	217 h 20 min	-
	Mémoire (Mo)	2 099	2 099	2 099	-
LoRDEC	Nombre de bases	60 892 408	77 969 503	188 228 237	1 154 508 245
	Taux d'erreurs (%)	0,0712	0,1474	0,5400	1,2643
	Rappel (%)	99,9921	99,9890	99,9483	99,8871
	Précision (%)	99,9313	99,8570	99,4730	98,7542
	Reads trimmés ou *splittés*	2 951	4 786	22 470	210 051
	Reads étendus	0	0	3	30
	Temps	6 min	8 min	28 min	6 h 01 min
	Mémoire (Mo)	438	455	799	2 238
FMLRC	Nombre de bases	64 715 552	83 732 334	223 529 739	1 832 773 801
	Taux d'erreurs (%)	0,1194	0,1930	1,0016	3,3582
	Rappel (%)	99,9142	99,8498	99,1797	97,8709
	Précision (%)	99,8815	99,8081	99,0020	96,6643
	Reads trimmés ou *splittés*	20	28	114	3 090
	Reads étendus	0	0	9	295
	Temps	23 min	29 min	1 h 19 min	8 h 02 min
	Mémoire (Mo)	387	408	906	7 939
HG-CoLoR	Nombre de bases	65 065 102	84 089 814	219 744 436	1 726 223 265
	Taux d'erreurs (%)	0,0430	0,0691	0,2959	0,6524
	Rappel (%)	99,9991	99,9982	99,9900	99,9682
	Précision (%)	99,9574	99,9315	99,7071	99,3554
	Reads trimmés ou *splittés*	133	498	4 562	71 079
	Reads étendus	5 969	8 180	19 237	140 791
	Temps	51 min	51 min	4 h 55 min	88 h 10 min
	Mémoire (Mo)	1 432	1 517	3 237	19 730

TABLE 5.4 – **Comparaison de la qualité de la correction fournie par HG-CoLoR et par les outils de correction de l'état de l'art, sur des *reads* simulés PacBio.** NaS n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. Daccord n'a pas pu corriger le jeu de données *C. elegans*, DALIGNER nécessitant plus de 250 Go de RAM lors de l'étape d'alignement.

5.4.2 Comparaison à l'état de l'art sur données simulées

Les différents outils de correction mentionnés précédemment ont tout d'abord été évalués sur les jeux de données simulées PacBio. De plus, pour l'intégralité des outils le permettant (Daccord, MECAT, CoLoRMap, LoRDEC, et HG-CoLoR) les versions *splittées* des *reads* longs corrigés ont été évaluées, afin de pouvoir étudier les proportions de *reads splittés* par les différents outils. Les *reads* longs corrigés par NaS et FMLRC n'ont cependant pas pu être évalués de cette façon, ces outils ne proposant pas d'option permettant de produire les *reads* longs corrigés en version *trimmée* ou *splittée*, et ne représentant pas les bases corrigées dans une casse différente des bases non corrigées.

Les statistiques décrivant la qualité de la correction de chacun de ces outils ont alors été obtenues en utilisant le premier module d'ELECTOR. Ces résultats, ainsi que le temps d'exécution et la consommation de mémoire des différents outils, sont présentés Table 5.4. Seules les principales statistiques reportées par ELECTOR sont présentées ici. L'ensemble des statistiques reportées par ELECTOR, pour les jeux de données simulées étudiés ici, est présenté dans le *benchmark* proposé Section 7.2.1.

Sur tous les jeux de données, MECAT s'est montré plus performant que tous les autres outils en termes de temps d'exécution. En particulier, la correction du jeu de données *C. elegans* a été réalisée par MECAT en moins de 20 minutes, soit plus rapidement que la correction du jeu de données *A. baylyi* par la majorité des autres outils, Daccord et LoRDEC exclus. La qualité de la correction proposée par MECAT s'est également révélée satisfaisante sur l'intégralité des jeux de données, les *reads* longs corrigés affichant, au plus, un taux d'erreurs de 0,6555% sur le jeu de données *S. cerevisiae*. De façon similaire, le rappel et la précision de MECAT ont atteint des valeurs supérieures à 99,3597% sur tous les jeux de données. Cependant, une large proportion de *reads* ont été *trimmés* ou *splittés* par la correction, et pour tous les jeux de données, le nombre de bases des *reads* corrigés par MECAT s'est montré plus faible que le nombre de bases des *reads* corrigés par les autres méthodes, sauf pour CoLoRMap sur le jeu de données *C. elegans*.

Le second outil d'auto-correction évalué, Daccord, s'est également montré plus performant en termes de temps d'exécution que l'intégralité des outils de correction hybride, LoRDEC exclus. Les taux d'erreurs des *reads* corrigés, atteignant au plus 0,5447%, ainsi que le rappel et la précision, atteignant respectivement au moins 99,4630%, et au moins 99,8591%, se sont également révélés satisfaisants, surpassant légèrement les valeurs reportées pour MECAT. Le nombre de *reads trimmés* ou *splittés* s'est également montré plus faible que celui de la plupart des autres méthodes, FMLRC exclus, ce dernier ne produisant pas de *reads trimmés* ou *splittés*. De la même façon, le nombre de bases des *reads* corrigés s'est montré plus important que le nombre de bases des *reads* corrigés par les autres méthodes, sauf pour FMLRC sur le jeu de données *S. cerevisiae*. La consommation de mémoire de Daccord s'est cependant révélée très élevée. En particulier, sur le jeu de données *C. elegans*, la première étape d'alignement des *reads* longs n'a pas pu être effectuée, DALIGNER nécessitant une quantité de mémoire bien trop importante. En effet, même sur un nœud de calcul disposant de 250 Go de RAM, DALIGNER n'a pas pu être lancé sur ce jeu de données. Ces importants besoins en termes de mémoire soulignent ainsi la grande difficulté de DALIGNER, et donc de Daccord, à passer à l'échelle sur des génomes de taille importante.

Les temps d'exécution de NaS se sont montrés particulièrement élevés, plus que l'intégralité des autres méthodes évaluées, en atteignant notamment plus de

24 heures sur le jeu de données *A. baylyi*. De plus, les taux d'erreurs des *reads* corrigés par NaS se sont également révélés plus élevés que les taux d'erreurs des *reads* corrigés par l'intégralité des autres méthodes. L'observation du rappel, atteignant plus de 99,9077%, et de la précision, atteignant au plus 97,2615%, sur tous les jeux de données, révèle que NaS tend à introduire des erreurs additionnelles lors de la correction. Ce comportement s'explique par le fait que NaS produit des *reads* longs corrigés à partir d'assemblages de *reads* courts, et peut donc sélectionner de mauvais *reads* lors de l'étape de recrutement, par exemple à cause de mauvaises identifications de copies des répétitions. De tels comportements résultent ainsi en des *reads* longs corrigés ne correspondant pas toujours aux *reads* longs originaux. Ce processus de recrutement et d'assemblage explique également la large proportion de *reads* étendus observée, des *reads* courts situés en dehors des *reads* longs originaux pouvant être recrutés et inclus dans l'assemblage.

CoLoRMap s'est montré efficace sur les jeux de données *A. baylyi*, *E. coli* et *S. cerevisiae*, les *reads* longs corrigés atteignant des taux d'erreurs inférieurs à 0,2655%, et le rappel et la précision atteignant respectivement au moins 99,9805% et au moins 99,74143%. En revanche, sur le jeu de données *C. elegans*, le taux d'erreurs des *reads* longs corrigés a atteint 2,6255%, le rappel restant relativement stable, et atteignant 99,8554%, mais la précision chutant à seulement 97,4526%. Ces valeurs soulignent ainsi une bonne identification des erreurs dans les *reads*, mais également l'introduction d'erreurs additionnelles lors de la correction. De plus, sur ce jeu de données, le nombre de bases des *reads* corrigés par CoLoRMap s'est montré plus faible que le nombre de bases des *reads* corrigés par toutes les autres méthodes. Cette chute de performance s'explique par la complexité du génome et par l'approche de correction adoptée par CoLoRMap, les *reads* courts étant plus difficiles à aligner sur les copies correctes des répétions dans le cas de génomes longs et complexes. Ces difficultés provoquent ainsi des manques de couverture sur certaines régions des *reads* longs, ou des corrections erronées, dû à des mauvaises identifications. Ces difficultés expliquent également l'importante proportion de *reads trimmés* ou *splittés* observée pour tous les jeux de données. Comparé à tous les autres outils, excepté NaS, CoLoRMap a également affiché les temps d'exécution les plus élevés. De la même façon, la consommation de mémoire de CoLoRMap s'est montrée plus importante que celle de tous les autres outils, excepté Daccord sur le jeu de données *S. cerevisiae*.

LoRDEC s'est montré efficace sur les jeux de données *A. baylyi* et *E. coli*, les *reads* longs corrigés atteignant des taux d'erreurs inférieurs à 0,1474%, et le rappel et la précision atteignant, respectivement, au moins 99,9890% et au moins 99,8570%. Cette efficacité tend cependant à s'atténuer à mesure que la complexité du génome augmente, le taux d'erreurs des *reads* longs corrigés atteignant 0,5400% sur le jeu de données *S. cerevisiae*, et jusqu'à 1,2643% sur le jeu de données *C. elegans*. Cette dégradation progressive de la qualité de la correction s'explique par la méthodologie adoptée par LoRDEC, utilisant un graphe de de Bruijn d'ordre fixe, et des k-mers solides des *reads* longs comme points d'ancrage sur ce graphe. Cette approche complexifie en effet les traversées et les recherches de chemins dans le cas de génomes complexes, notamment au sein des régions répétées, des k-mers contenant des erreurs ayant alors de grandes chances d'être considérés comme solides et utilisés comme ancres. De ce fait, une très large proportion de *reads* longs corrigés *trimmés* ou *splittés* peut être observée, la plus large proportion parmi toutes les méthodes de correction hybride. En revanche, LoRDEC s'est révélé comme l'outil de correction hybride le plus rapide, parmi tous les outils évalués, et s'est également montré hautement efficace en termes de consommation de mémoire, demandant à peine plus de 2 Go pour la correction de jeu de données *C. elegans*.

FMLRC, de son côté, a réussi à corriger efficacement les jeux de données *A. baylyi* et *E. coli*, les *reads* longs corrigés affichant des taux d'erreurs d'au plus 0,1930%, et le rappel et la précision atteignant, respectivement, au moins 99,8498% et au moins 99,8081%. Sur les jeux de données restants, le taux d'erreurs des *reads* longs corrigés a augmenté, atteignant jusqu'à 3,3582% sur le jeu de données *C. elegans*, et le rappel et la précision ont tous deux diminué, atteignant respectivement 97,8709% et 96,6443% sur ce même jeu de données. Cette diminution de la qualité des résultats s'explique par le fait que FMLRC ne propose pas de *trimmer* ou de *splitter* les *reads*, et ne reporte pas les bases corrigées dans une casse différente des bases non corrigées. Ainsi, l'évaluation des *reads* corrigés par FMLRC a également pris en compte les bases non corrigées des *reads* longs. Cependant, cette baisse de qualité des résultats souligne également une difficulté à passer à l'échelle sur des génomes longs et complexes, malgré l'utilisation d'un graphe de de Bruijn d'ordre variable. La limitation de ce graphe à une correction en deux itérations, utilisant deux valeurs de k différentes, peut cependant expliquer la difficulté de passage à l'échelle, de hautes valeurs de k étant parfois nécessaires afin de résoudre efficacement les régions répétées lors des traversées du graphe. Malgré cette double passe de correction, les temps d'exécution de FMLRC se sont tout de même révélés comparables à ceux de LoRDEC, ne demandant que 2 heures supplémentaires pour la correction du jeu de données *C. elegans*. La consommation de mémoire s'est également montrée comparable entre les deux outils, sauf sur ce même jeu de données, FMLRC consommant quasiment quatre fois plus de mémoire que LoRDEC.

Sur tous les jeux de données, les *reads* corrigés par HG-CoLoR ont affiché les plus faibles taux d'erreurs, atteignant un minimum de 0,0430% sur le jeu de données *A. baylyi*, et un maximum de 0,6524% sur le jeu de données *C. elegans*. De la même façon, le rappel de HG-CoLoR s'est révélé plus élevé que celui de tous les autres outils, sur tous les jeux de données, et la précision s'est également révélée plus élevée que celle de tous les autres outils, excepté CoLoRMap sur le jeu de données *S. cerevisiae* et MECAT sur le jeu de données *C. elegans*. Le nombre de bases des *reads* corrigés par HG-CoLoR a également surpassé le nombre de bases des *reads* corrigés par tous les autres outils, sauf pour Daccord sur le jeu de données *S. cerevisiae* et pour FMLRC sur les jeux de données *S. cerevisiae* et *C. elegans*. L'avantage de ce dernier est cependant lié au fait qu'il ne propose pas de *trimmer* ou de *splitter* les *reads*, contrairement à HG-CoLoR. La proportion de *reads trimmés* ou *splittés* s'est également révélée 3 à 22 fois plus faible que la proportion de *reads trimmés* ou *splittés* par LoRDEC, soulignant ainsi l'intérêt du graphe de de Bruijn d'ordre variable. La nette augmentation de la qualité des *reads* longs corrigés par rapport à FMLRC souligne également l'intérêt de ne pas limiter l'utilisation du graphe à un nombre fini de valeurs de k. La proportion de *reads* étendus par HG-CoLoR s'est cependant montrée bien plus importante que la proportion de *reads* étendus par LoRDEC ou par FMLRC. Bien que les trois méthodes utilisent des approches similaires pour la correction des extrémités des *reads* longs, ces observations sont cependant cohérentes avec la méthodologie adoptée par HG-CoLoR. En effet, contrairement à LoRDEC et à FMLRC, HG-CoLoR ne propose pas de réaligner les extrémités corrigées sur les *reads* longs originaux afin de déterminer le préfixe / le suffixe optimal à utiliser comme correction. Ainsi, HG-CoLoR se base uniquement sur la longueur des extrémités à corriger, et sur la longueur des chemins uniques pouvant être suivis au sein du graphe. En termes de temps d'exécution, bien que plus coûteux que LoRDEC et FMLRC, HG-CoLoR s'est montré comparable à CoLoRMap. La même remarque peut-être faite concernant la consommation de mémoire de HG-CoLoR par rapport à LoRDEC et FMLRC, HG-CoLoR étant cependant moins coûteux que CoLoRMap et que Daccord. L'ensemble

de ces observations soulignent ainsi que HG-CoLoR fournit le meilleur compromis entre qualité des résultats et consommation de ressources, sur ces jeux de données simulées, et est le seul outil capable de passer efficacement à l'échelle sur le génome eucaryote de *C. elegans*.

5.4.3 Comparaison à l'état de l'art sur données réelles

Les différents outils de correction mentionnés précédemment ont ensuite été évalués sur les jeux de données réelles ONT. Pour ces jeux de données, la qualité des *reads* corrigés ainsi que la qualité des assemblages pouvant être générés à partir de ces derniers sont évaluées. Au vu des importants taux d'erreurs des *reads* ONT composant ces jeux de données (de 20 à 44% d'erreurs), le second module d'ELECTOR a été utilisé afin d'obtenir les statistiques décrivant la qualité des *reads* longs corrigés par chacun des outils. Utiliser le premier module d'ELECTOR en mode données réelles n'aurait en effet pas permis d'obtenir des résultats pertinents, les *reads* non corrigés affichant des taux d'erreurs trop importants. Ces taux d'erreurs auraient alors introduit des biais lors de l'étape d'alignement permettant de déterminer les *reads* de référence, et donc biais dans l'évaluation. De plus, ce second module permet également d'évaluer la qualité des assemblages pouvant être générés à partir des *reads* longs corrigés.

Comme pour les expériences sur données simulées, pour les méthodes le permettant, les versions *splittées* des *reads* ont été évaluées lors de l'étude de la qualité des *reads* longs corrigés, encore une fois, afin de permettre l'étude des proportions de *reads splittés* par les différents outils. Pour l'étude de la qualité des assemblages, en revanche, les assemblages ont été réalisés à partir des versions non *trimmées* et non *splittées* des *reads*. Ce choix a été réalisé afin de permettre à l'assembleur d'utiliser pleinement l'information portée par la longueur des *reads*, et ainsi éviter les biais liés aux grandes proportions de *reads splittés* ayant été observées pour les différents outils lors des expériences sur données simulées, et pouvant affecter la qualité des assemblages.

5.4.3.1 Qualité des *reads* longs corrigés

Les statistiques reportées par le second module d'ELECTOR, pour la partie décrivant la qualité des *reads* longs corrigés, ainsi que le temps d'exécution et la consommation de mémoire des différents outils de correction, sont présentés Table 5.5.

Malgré l'importante profondeur de séquençage de ces jeux de données réelles, Daccord n'est pas parvenu à corriger les *reads* longs de manière efficace, ceux-ci s'alignant avec une identité moyenne d'au plus 96,7690% pour le jeu de données *E. coli*. De plus, les difficultés de passage à l'échelle de Daccord l'ont empêché de corriger les jeux données *S. cerevisiae* et *C. elegans*, l'étape d'alignement avec DALIGNER nécessitant une nouvelle fois une quantité déraisonnable de mémoire.

De la même façon, MECAT n'est pas non plus parvenu à corriger efficacement les *reads* longs, ceux-ci s'alignant cette fois avec une identité moyenne d'au plus 92,2218% pour le jeu de données *E. coli*, et atteignant une identité moyenne aussi faible que 80,0763% pour le jeu de données *S. cerevisiae*. De plus, la correction n'a pas pu être réalisée sur le jeu de données *C. elegans*, MECAT s'arrêtant en signalant une erreur lors de l'étape d'alignement.

Les résultats fournis par ces deux outils tendent ainsi à montrer que, malgré des profondeurs de séquençage importantes, l'auto-correction demeure insatisfaisante

lorsqu'appliquée à des *reads* longs extrêmement bruités, tels que les *reads* ONT issus des premières expériences de séquençage de cette technologie.

Concernant NaS, l'identité moyenne des *reads* longs corrigés a atteint au minimum 99,8995% sur le jeu de données *S. cerevisiae*. Ces résultats diffèrent grandement des statistiques observées pour NaS lors des expériences réalisées sur données simulées. Comme expliqué lors de l'analyse de ces expériences, Section 5.4.2, ces différences sont liées à la méthodologie de NaS, pouvant recruter et assembler des *reads* courts provenant de copies différentes des répétitions. Ces assemblages peuvent ainsi résulter en des *reads* longs corrigés de haute qualité, ne s'alignant pas parfaitement avec leurs *reads* longs originaux, mais s'alignant plutôt avec une haute identité sur une autre région du génome. Le temps d'exécution de NaS a également augmenté en fonction de la taille du génome et de la profondeur de séquençage des *reads* longs, comme observé lors des expériences réalisées sur données simulées. En particulier, la correction du jeu de données *S. cerevisiae* a demandé plus de 16 jours, et la correction du jeu de données *C. elegans* n'a donc pas été réalisée.

Les statistiques reportées pour CoLoRMap sont cohérentes avec celles observées lors des expériences sur données simulées. CoLoRMap est ainsi parvenu à corriger les *reads* longs de manière satisfaisante, et ce jusqu'au jeu de données *S. cerevisiae*, malgré son important taux d'erreurs. Une importante proportion de *reads* a cependant été *splittée*, comme le soulignent les métriques décrivant la longueur moyenne des *reads*. Sur le jeu de données *C. elegans*, la qualité de la correction a légèrement diminué, comme lors de l'expérience sur les données simulées provenant de ce même génome. L'identité moyenne des *reads* longs corrigés a ainsi atteint 98,6614%, et les alignements n'ont couvert que 96,4079% du génome.

Les mêmes remarques peuvent également être appliquées à LoRDEC, proposant une correction satisfaisante jusqu'au jeu de données *E. coli*, la qualité diminuant ensuite légèrement, comme lors des expériences réalisées sur données simulées. En particulier, l'identité moyenne des *reads* longs corrigés a atteint 98,2718% sur le jeu de données *C. elegans*. De plus, LoRDEC s'est arrêté en reportant une erreur signalant un *read* trop long, lors de la correction de ce jeu de données. Toujours comme observé lors des expériences sur données simulées, LoRDEC a conservé sa tendance à *splitter* une large proportion des *reads* longs corrigés, comme le soulignent les métriques décrivant la longueur moyenne des *reads*.

Sur tous les deux de données, FMLRC est parvenu à produire une version corrigée de chaque *read* long original. Ainsi, le nombre de bases des *reads* longs corrigés par FMLRC s'est montré plus important que le nombre de bases des *reads* longs corrigés par les autres méthodes. Cependant, seule une faible proportion de ces *reads* a pu être alignée. En effet, FMLRC affiche la plus faible proportion de *reads* corrigés alignés, parmi tous les outils de correction évalués, et ce sur tous les jeux de données. La plus faible proportion est notamment atteinte sur le jeu de données *A. baylyi*, où seulement 27,7741% des *reads* corrigés ont pu être alignés. La qualité des *reads* corrigés ayant pu être alignés est cependant cohérente avec les observations réalisées lors des expériences sur données simulées. FMLRC est en effet parvenu à corriger efficacement les jeux de données *A. baylyi* et *E. coli*, mais a rencontré plus de difficulté avec les deux autres jeux de données, l'identité moyenne des *reads* alignés atteignant seulement 96,5611% sur le jeu de données *C. elegans*.

À l'exception de NaS, HG-CoLoR a produit les *reads* longs corrigés s'alignant avec la plus haute identité, atteignant au minimum 99,5430% sur le jeu de données *C. elegans*. Comme observé lors de l'expérience sur données simulées, et comme le soulignent les métriques décrivant la longueur moyenne des *reads*, HG-CoLoR a permis de lever les limitations de LoRDEC liées à l'utilisation d'un graphe de de Bruijn

d'ordre fixe, tout en proposant des corrections de meilleure qualité que FMLRC. Le nombre de bases des *reads* corrigés par HG-CoLoR s'est également révélé plus élevé que le nombre de bases des *reads* corrigés par l'intégralité des autres méthodes, excepté NaS sur le jeu de données *E. coli*, et excepté FMLRC. Ce dernier n'est cependant pas comparable, au vu des faibles proportions de *reads* corrigés pouvant être alignés, et au vu du fait que les *reads* corrigés qu'il produit ne puissent être *splittés*.

Le temps d'exécution et la consommation de mémoire de HG-CoLoR se sont encore une fois montrés plus élevés que ceux de LoRDEC et de FMLRC, mais plus faibles que ceux de CoLoRMap. Ainsi, sur ces jeux de données réelles, HG-CoLoR fournit encore une fois le meilleur compromis entre qualité des résultats et consommation de ressources. Ces résultats confirment également la capacité de HG-CoLoR à passer à l'échelle sur de plus larges génomes eucaryotes, tels que celui de *C. elegans*, même sur des données réelles ONT affichant de plus importants taux d'erreurs.

5.4.3.2 Qualité des assemblages

Les statistiques reportées par le second module d'ELECTOR, pour la partie décrivant la qualité des assemblages générés à partir des *reads* longs corrigés, sont présentées Table 5.6.

Malgré les importants taux d'erreurs observés dans la Table 5.5, les *reads* longs corrigés par les deux méthodes d'auto-correction sont parvenus à s'assembler de manière satisfaisante pour le jeu de données *A. baylyi*. Les *reads* longs corrigés par Daccord ont en effet produit deux contigs, l'un d'eux représentant l'intégralité du génome, tandis que les *reads* longs corrigés par MECAT ont produit un unique contig, cependant légèrement plus court que le génome de référence. Sur le jeu de données *E. coli*, en revanche, ces deux méthodes ne sont pas parvenues à produire des *reads* longs corrigés pouvant générer des assemblages de haute qualité. L'assemblage généré à partir des *reads* longs corrigés par Daccord est cependant légèrement meilleur que celui généré à partir des *reads* longs corrigés par MECAT, car composé d'un contig de moins, et affichant des tailles NGA50 et NGA75 plus importantes. Ces observations sont cependant cohérentes avec la profondeur de séquençage des *reads* longs du jeu de données *E. coli*, atteignant seulement 29x, contre 106x pour le jeu de données *A. baylyi*. La qualité de l'assemblage de ce jeu de données peut en effet s'expliquer par l'importante profondeur de séquençage des *reads*. Les difficultés des méthodes d'auto-correction à produire des *reads* longs corrigés pouvant générer des assemblages de haute qualité, sur des jeux de données disposant de profondeurs de séquençage importantes, mais affichant des taux d'erreurs élevés, se confirment cependant avec les résultats d'assemblage des *reads* du jeu de données *S. cerevisiae*, après correction par MECAT. En effet, l'assemblage généré ici ne permet de couvrir que 15,1658% du génome de référence, et affiche des tailles NGA50 et NGA75 plus courtes que la longueur moyenne des *reads* longs du jeu de données initial. Ces observations sont encore une fois cohérentes avec le taux d'erreurs de près de 20%, et la faible proportion de *reads* longs corrigés par MECAT, décrits dans la Table 5.5 pour ce jeu de données.

Sur le jeu de données *A. baylyi*, bien que proposant des corrections de meilleure qualité que les deux méthodes d'auto-correction, les *reads* longs corrigés par CoLoRMap, FMLRC et LoRDEC ont généré des assemblages de plus mauvaise qualité. Pour CoLoRMap et FMLRC, bien que les *reads* corrigés aient pu s'assembler en un contig unique, ce dernier n'est pas parvenu à couvrir l'intégralité du génome du référence. Pour LoRDEC, les *reads* corrigés ont produit un assemblage composé de

deux contigs, contenant tous deux une part non négligeable de l'information du génome de référence, comme l'indiquent les tailles NGA50 et NGA75. De plus, cet assemblage n'a permis de couvrir que 94,6755% du génome de référence, soit la plus faible couverture de tous les assemblages, pour ce jeu de données. L'assemblage généré à partir des *reads* longs corrigés par HG-CoLoR, bien qu'également composé de deux contigs, contient cependant l'intégralité de l'information du génome de référence dans un seul de ces contigs, comme l'indiquent les tailles NGA50 et NGA75, atteignant la taille du génome de référence. De plus, contrairement aux méthodes de correction hybride citées précédemment, l'assemblage généré à partir des *reads* longs corrigés par HG-CoLoR permet de couvrir l'intégralité du génome. Sur ce jeu de données, les *reads* longs corrigés par NaS ont également permis de générer un assemblage composé d'un unique contig, couvrant l'intégralité du génome de référence.

Sur le jeu de données *E. coli*, les *reads* longs corrigés par CoLoRMap et LoRDEC ont généré des assemblages comparables, composés d'un contig unique, et affichant des tailles NGA50 et NGA75 correspondant à la taille du génome de référence, mais ne couvrant pas l'intégralité de celui-ci. Outre ces deux méthodes, les *reads* longs corrigés par les autres méthodes de correction hybride, NaS, FMLRC et HG-CoLoR, ont tous permis de générer des assemblages composés d'un contig unique et couvrant l'intégralité du génome de référence.

Sur le jeu de données *S. cerevisiae*, malgré les observations précédentes, les *reads* longs corrigés par LoRDEC ont permis de générer un assemblage de meilleure qualité que les *reads* longs corrigés par CoLoRMap, et même par NaS, générant pourtant des assemblages de très haute qualité sur les deux jeux de données précédents. Cet assemblage, composé de moins de contigs, a en effet affiché des tailles NGA50 et NGA75 plus élevées, et a également permis de couvrir une plus large proportion du génome de référence. En revanche, sur le jeu de données *C. elegans*, les *reads* longs corrigés par LoRDEC ont généré un assemblage de très mauvaise qualité, composé d'un très faible nombre de contigs très courts, et couvrant moins de 0,1% du génome de référence. Ce mauvais assemblage peut cependant s'expliquer par le fait que LoRDEC ne soit pas parvenu à corriger l'intégralité du jeu de données, et se soit arrêté en reportant une erreur durant la correction. Ces résultats soulignent cependant la difficulté de LoRDEC à passer efficacement à l'échelle sur des génomes longs et complexes, notamment à cause de l'utilisation d'un graphe de de Bruijn d'ordre fixe.

En effet, FMLRC et HG-CoLoR, utilisant tous deux un graphe de de Bruijn d'ordre variable, ont permis de produire les *reads* longs corrigés générant les meilleurs assemblages sur les jeux de données *S. cerevisiae* et *C. elegans*. En particulier, sur le jeu de données *S. cerevisiae*, affichant à la base un taux d'erreurs de plus de 44%, les *reads* longs corrigés par HG-CoLoR ont même permis de générer un assemblage de meilleure qualité que les *reads* longs corrigés par FMLRC. Cet assemblage est en effet composé d'un contig de moins, affiche des tailles NGA50 et NGA75 plus élevées, et couvre même plus de 3% de bases supplémentaires du génome de référence. Ces résultats soulignent encore une fois l'intérêt de ne pas limiter l'utilisation du graphe de de Bruijn d'ordre variable à un nombre fini de valeurs de k. De la même façon, pour le jeu de données *C. elegans*, les *reads* longs corrigés par HG-CoLoR ont résulté en un assemblage composé de contigs moins nombreux, et affichant des tailles NGA50 et NGA75 plus élevées que l'assemblage généré à partir des *reads* longs corrigés par FMLRC. Ce dernier a cependant permis de couvrir une proportion légèrement plus importante (0,2881%) du génome de référence.

Sur ce même jeu de données *C. elegans*, les *reads* longs corrigés par CoLoRMap

ont permis de générer un assemblage de bien meilleure qualité que les *reads* longs corrigés par LoRDEC, couvrant 76,6546% du génome. Ces résultats soulignent ainsi une meilleure capacité de passage à l'échelle sur des génomes longs et complexes, bien que seulement la moitié des contigs produits aient pu être alignés sur le génome de référence. Cependant, la qualité de cet assemblage, plus faible que celle des assemblages générés à partir des *reads* longs corrigés par FMLRC ou par HG-CoLoR, souligne également l'intérêt d'utiliser un graphe de de Bruijn d'ordre variable pour la correction hybride, plutôt que de se baser uniquement sur l'alignement des *reads* courts sur les *reads* longs.

5.4.4 Origine des bases dans les *reads* longs corrigés

Des descriptions indiquant les proportions de bases des *reads* longs corrigés provenant des graines, des traversées du graphe, et des *reads* long originaux, pour l'intégralité des jeux de données étudiés, sont présentées Table 5.7. Ces descriptions montrent que, pour les jeux de données simulées PacBio, affichant des taux d'erreurs de 18,6%, la plupart des bases des *reads* corrigés proviennent des graines. Les taux d'erreurs de ces *reads* longs simulés étant relativement faibles par rapport aux taux d'erreurs des *reads* longs réels ONT, les *reads* courts ont effectivement pu s'aligner plus facilement. De plus larges régions des *reads* longs ont ainsi pu être couvertes, et moins de traversées du graphe ont alors été nécessaires à la correction. Sur ces jeux de données simulées, seule une très faible quantité de bases non corrigées, provenant des *reads* longs originaux, peut être observée, la plus large proportion étant d'à peine plus de 1%, pour le jeu de données *S. cerevisiae*. Ces observations s'appliquent également au jeu de données réelles *E. coli*, pour lequel le taux d'erreurs des *reads* longs originaux n'est que d'un peu plus de 20%.

Sur les trois autres jeux de données réelles ONT, affichant des taux d'erreurs de près de 30 à 44%, une plus large proportion des bases des *reads* corrigés proviennent des traversées du graphe. Par exemple, sur le jeu de données *C. elegans*, affichant un taux d'erreurs de 29%, presque autant de bases proviennent des traversées du graphe que des graines. Sur le jeu de données *S. cerevisiae*, affichant un taux d'erreurs de plus de 44%, plus de deux fois plus de bases proviennent des traversées du graphe que des graines. Cette large proportion de bases indique que l'utilisation du graphe permet de corriger efficacement les régions extrêmement bruitées des *reads* longs, sur lesquelles l'alignement est impossible.

De manière plus générale, la proportion non négligeable de bases provenant des traversées du graphe, sur l'ensemble des jeux de données, réelles comme simulées, confirme l'intérêt du graphe, et sa capacité à corriger les régions des *reads* longs n'ayant pu être couvertes lors de l'étape d'alignement.

La proportion de bases non corrigées, provenant des *reads* longs originaux, tend également à augmenter en fonction du taux d'erreurs des *reads* longs, et de la complexité du génome étudié. En effet, la plus large proportion de bases non corrigées, de plus de 15%, peut être observée sur le jeu de données réelles *S. cerevisiae*, affichant le plus haut taux d'erreurs parmi tous les jeux de données considérés. Cependant, sur le jeu de données réelles *C. elegans*, affichant un taux d'erreurs légèrement plus faible que le jeu de données réelles *A. baylyi*, presque 7% des bases des *reads* longs corrigés proviennent des *reads* longs originaux, et n'ont donc pas pu être corrigées. De même, sur le jeu de données simulées *C. elegans*, affichant un taux d'erreurs identique aux autres jeux de données simulées, près de 4% des bases des *reads* longs corrigés proviennent des *reads* longs originaux, et n'ont donc pas pu être corrigées. Ces

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Daccord	Nombre de *reads*	53 926	26 502	-	-
	Nombre de bases	174 962 080	119 130 415	-	-
	Longueur moyenne (bp)	3 244	4 495	-	-
	Reads alignés (%)	97,1906	99,9170	-	-
	Identité moyenne (%)	93,2546	96,7690	-	-
	Couverture du génome (%)	100,0000	100,0000	-	-
	Temps	43 min	20 min	-	-
	Mémoire (Mo)	25 801	6 515	-	-
MECAT	Nombre de *reads*	16 811	16 089	14 740	-
	Nombre de bases	154 436 057	106 757 446	83 553 890	-
	Longueur moyenne (bp)	9 186	6 635	5 668	-
	Reads alignés (%)	100,0000	100,0000	99,4573	-
	Identité moyenne (%)	91,4676	92,2218	80,0763	-
	Couverture du génome (%)	100,0000	100,0000	92,6533	-
	Temps	23 min	4 min	14 min	-
	Mémoire (Mo)	2 978	1 681	7 374	-
CoLoRMap	Nombre de *reads*	36 422	25 635	72 017	184 092
	Nombre de bases	141 415 030	114 722 711	165 218 405	418 509 369
	Longueur moyenne (bp)	3 882	4 475	2 294	2 273
	Reads alignés (%)	99,9973	100,0000	99,7403	99,9473
	Identité moyenne (%)	99,5079	99,5751	99,6958	98,6614
	Couverture du génome (%)	100,0000	100,0000	99,1528	96,4079
	Temps	3 h 41 min	2 h 01 min	10 h 44 min	91 h 18 min
	Mémoire (Mo)	13 028	12 121	18 241	31 349
NaS	Nombre de *reads*	24 063	21 818	71 793	-
	Nombre de bases	212 707 189	172 918 739	426 326 355	-
	Longueur moyenne (bp)	8 839	7 925	5 938	-
	Reads alignés (%)	100,0000	100,0000	99,8231	-
	Identité moyenne (%)	99,9900	99,9793	99,8985	-
	Couverture du génome (%)	100,0000	100,0000	98,7695	-
	Temps	94 h 19 min	72 h 02 min	> 390 h[1]	-
	Mémoire (Mo)	2 099	2 099	-[1]	-
LoRDEC	Nombre de *reads*	50 776	31 728	196 091	167 699
	Nombre de bases	175 140 999	125 542 707	220 706 773	221 676 779
	Longueur moyenne (bp)	3 449	3 956	1 125	1 321
	Reads alignés (%)	99,8779	96,6118	98,5094	96,0137
	Identité moyenne (%)	99,9448	99,9300	98,8168	98,2718
	Couverture du génome (%)	100,0000	100,0000	98,8934	85,0782
	Temps	16 min	13 min	1 h 09 min	1 h 15 min
	Mémoire (Mo)	436	458	797	2 373
FMLRC	Nombre de *reads*	89 011	22 270	205 923	363 500
	Nombre de bases	390 735 419	134 402 291	1 185 455 434	2 063 059 647
	Longueur moyenne (bp)	4 389	6 035	5 756	5 675
	Reads alignés (%)	27,7741	98,2937	31,7497	78,1986
	Identité moyenne (%)	99,6779	99,9252	96,7164	96,5611
	Couverture du génome (%)	100,0000	100,0000	99,8120	99,9932
	Temps	2 h 01 min	43 min	6 h 15 min	7 h 54 min
	Mémoire (Mo)	449	384	876	7 141
HG-CoLoR	Nombre de *reads*	25 536	21 986	76 193	305 777
	Nombre de bases	284 883 716	133 956 298	512 438 767	1 567 516 029
	Longueur moyenne (bp)	11 156	6 092	6 725	5 126
	Reads alignés (%)	99,9883	100,0000	99,7126	99,8653
	Identité moyenne (%)	99,9760	99,9589	99,7176	99,5430
	Couverture du génome (%)	100,0000	100,0000	99,5341	99,9655
	Temps	1 h 34 min	59 min	8 h 51 min	83 h 10 min
	Mémoire (Mo)	3 750	1 508	11 575	19 836

TABLE 5.5 – **Comparaison de la qualité des *reads* longs corrigés par HG-CoLoR et par les outils de correction de l'état de l'art, sur des *reads* réels ONT.** Daccord n'a pas pu corriger les jeux de données *S. cerevisiae* et *C. elegans*, l'étape d'alignement avec DALIGNER nécessitant plus de 250 Go de RAM. MECAT n'a pas pu corriger le jeu de données *C. elegans*, l'étape d'alignement s'arrêtant en reportant une erreur. NaS n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. Les métriques décrivant le nombre de *reads* dénombrent les fragments produits.

[1] La correction du jeu de données *S. cerevisiae* par NaS n'a pas abouti après 16 jours. Les *reads* corrigés de ce jeu de données ont donc été obtenus sur le site du Génoscope : http://www.genoscope.cns.fr/externe/nas/datasets/NaS/yeast/

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Daccord	Nombre de contigs	2	6	-	-
	Nombre de contigs alignés	2	6	-	-
	Nombre de *breakpoints*	4	1	-	-
	NGA50 (bp)	3 472 403	1 140 945	-	-
	NGA75 (bp)	3 472 403	995 717	-	-
	Couverture du génome (%)	100,0000	99,9572	-	-
MECAT	Nombre de contigs	1	7	102	-
	Nombre de contigs alignés	1	7	102	-
	Nombre de *breakpoints*	3	1	2	-
	NGA50 (bp)	3 335 596	843 326	3 163	-
	NGA75 (bp)	3 335 596	610 826	3 163	-
	Couverture du génome (%)	99,9985	99,9204	15,1658	-
CoLoRMap	Nombre de contigs	1	1	71	1 246
	Nombre de contigs alignés	1	1	68	1 237
	Nombre de *breakpoints*	54	40	47	617
	NGA50 (bp)	3 588 032	4 657 185	204 052	58 411
	NGA75 (bp)	3 588 032	4 657 185	88 468	11 122
	Couverture du génome (%)	96,4454	98,5239	87,1140	76,6546
NaS	Nombre de contigs	1	1	159	-
	Nombre de contigs alignés	1	1	159	-
	Nombre de *breakpoints*	2	6	17	-
	NGA50 (bp)	3 600 775	4 641 680	85 205	-
	NGA75 (bp)	3 600 775	4 641 680	43 313	-
	Couverture du génome (%)	99,9986	99,9674	89,7248	-
LoRDEC	Nombre de contigs	2	1	66	21
	Nombre de contigs alignés	2	1	66	16
	Nombre de *breakpoints*	73	71	126	9
	NGA50 (bp)	3 010 005	4 659 557	361 726	589
	NGA75 (bp)	3 010 005	4 659 557	178 604	589
	Couverture du génome (%)	94,6755	97,7848	90,0706	0,0854
FMLRC	Nombre de contigs	1	1	54	1 047
	Nombre de contigs alignés	1	1	54	1 042
	Nombre de *breakpoints*	21	3	69	713
	NGA50 (bp)	3 673 304	4 642 134	378 469	91 545
	NGA75 (bp)	3 673 304	4 642 134	186 053	40 293
	Couverture du génome (%)	96,8804	100,0000	92,2118	82,9120
HG-CoLoR	Nombre de contigs	2	1	53	989
	Nombre de contigs alignés	2	1	53	988
	Nombre de *breakpoints*	5	5	72	461
	NGA50 (bp)	3 595 353	4 643 849	470 355	96 058
	NGA75 (bp)	3 595 353	4 643 849	212 761	41 829
	Couverture du génome (%)	99,9182	99,9739	95,3867	82,6239

TABLE 5.6 – **Comparaison de la qualité des assemblages générés à partir des *reads* longs corrigés par HG-CoLoR et par les outils de correction de l'état de l'art, sur des *reads* réels ONT.** Daccord n'a pas pu corriger les jeux de données *S. cerevisiae* et *C. elegans*, l'étape d'alignement avec DALIGNER nécessitant plus de 250 Go de RAM. MECAT n'a pas pu corriger le jeu de données *C. elegans*, l'étape d'alignement s'arrêtant en reportant une erreur. NaS n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. Les assemblages ne sont donc pas évalués pour ces outils sur ces jeux de données.

Jeu de données	Provenance des bases		
	Graines (%)	Graphe (%)	*Reads* originaux (%)
A. baylyi (simulé)	68,95	30,97	0,08
E. coli (simulé)	67,01	32,74	0,25
S. cerevisiae (simulé)	63,97	34,82	1,21
C. elegans (simulé)	58,94	37,07	3,99
A. baylyi (réel)	36,37	60,53	3,10
E. coli (réel)	62,61	37,04	0,35
S. cerevisiae (réel)	24,41	60,22	15,37
C. elegans (réel)	48,94	44,23	6,83

TABLE 5.7 – **Proportion des bases provenant des graines, des traversées du graphe, et des *reads* longs originaux, dans les *reads* longs corrigés par HG-CoLoR.**

proportions de bases non corrigées relativement importantes semblent donc provenir de la complexité du génome, et être dues au fait que les régions répétées n'ont pas toujours pu être résolues correctement, malgré l'utilisation d'un ordre maximal important lors de la construction du graphe.

5.5 Synthèse

Dans ce chapitre, nous avons décrit HG-CoLoR, une nouvelle méthode de correction hybride de *reads* longs, s'inspirant de NaS, et combinant deux approches de l'état de l'art, afin de lever ses difficultés de passage à l'échelle, notamment en termes de temps. Une stratégie d'alignement de *reads* courts est ainsi combinée à l'utilisation d'un graphe de de Bruijn, construit à partir des *reads* courts. Pour cela, HG-CoLoR adopte une stratégie de type *seed and extend*, où les *reads* courts sont d'abord alignés sur les *reads* longs, afin de mettre en évidence des graines. Ces graines sont ensuite utilisées comme points d'ancrage sur le graphe de de Bruijn, et sont alors reliées en y recherchant des chemins, afin de corriger les régions des *reads* longs non couvertes par les alignements initiaux.

De plus, le graphe de de Bruijn utilisé par HG-CoLoR a la particularité d'être d'ordre variable, et permet ainsi d'éviter les limitations intrinsèques des graphes de de Bruijn classiques, rencontrant des difficultés avec la résolution des régions répétées, ou comportant de nombreux embranchements complexes, en fonction de l'ordre choisi. Cependant, contrairement à FMLRC, utilisant également un graphe de de Bruijn d'ordre variable, HG-CoLoR ne limite pas l'utilisation de ce graphe à un nombre fini de valeurs de k. En effet, HG-CoLoR permet de parcourir l'ensemble des graphes de de Bruijn, entre un ordre minimal k et un ordre maximal K, afin d'assurer une correction de haute qualité, et de limiter les cas où aucun chemin ne peut être trouvé entre une paire d'ancres donnée.

Les expériences réalisées, aussi bien sur des données simulées PacBio que sur des données réelles ONT, montrent que HG-CoLoR fournit le meilleur compromis entre qualité de la correction et consommation de ressources, par rapport aux autres méthodes de l'état de l'art. La qualité de la correction fournie par HG-CoLoR induit également des assemblages de bonne qualité, globalement meilleurs que les assemblages pouvant être générés à partir des *reads* longs corrigés par les autres méthodes de l'état de l'art. En particulier, sur un jeu de données réelles ONT affichant un taux

d'erreurs de plus de 44%, HG-CoLoR a permis de produire des *reads* corrigés d'excellente qualité, affichant un taux d'erreurs de moins de 0,3%. Ces *reads* corrigés ont également permis de générer un assemblage de haute qualité, comportant moins de contigs et couvrant davantage le génome de référence que les assemblages générés à partir des *reads* corrigés par les autres méthodes. De plus, sur le génome eucaryote de *C. elegans*, aussi bien pour des données simulées PacBio que pour des données réelles ONT, la qualité de la correction produite par HG-CoLoR n'a que très légèrement diminué, contrairement aux autres méthodes, affichant de nettes difficultés de passage à l'échelle.

Les résultats de ces expériences soulignent également le fait que, même avec une importante profondeur de séquençage, les méthodes d'auto-correction ne permettent pas de corriger efficacement les *reads* longs disposant de taux d'erreurs très élevés, tels que les *reads* ONT séquencés à l'aide des premières versions des chimies. Ces observations confirment donc l'intérêt des méthodes de correction hybride pour la correction de ces *reads*.

Les résultats de ces expériences soulignent également le fait que, bien que capables de réaliser des corrections relativement satisfaisantes sur des jeux de données affichant des taux d'erreurs inférieurs à 20%, et disposant de faibles profondeurs de séquençage, les méthodes d'auto-correction ne permettent pas de traiter efficacement les *reads* longs disposant de taux d'erreurs plus élevés, tels que les *reads* ONT séquencés à l'aide des premières versions des chimies, même si la profondeur de séquençage de ces derniers est importante. Ces observations confirment donc l'intérêt des méthodes de correction hybride pour la correction de ces *reads*.

Chapitre 6

Auto-correction de données *ultra-long reads*

Sommaire

6.1 Introduction

6.1.1 Contexte

Les technologies de séquençage de troisième génération ont grandement évolué depuis leur introduction en 2011. En particulier, les taux d'erreurs des *reads*, atteignant 15 à 30% lors des premières expériences de séquençage, ont été grandement

réduits à mesure de ces évolutions. En effet, aussi bien pour PacBio que pour ONT, les expériences de séquençage récentes permettent de produire des *reads* affichant des taux d'erreurs compris entre 10 et 12% en moyenne. Cependant, même avec les développements les plus récents, ces taux d'erreurs semblent actuellement stagner aux alentours de ces valeurs, et ne plus afficher de nouvelle tendance à la diminution. Ainsi, bien qu'ayant grandement diminué, ces taux d'erreurs demeurent relativement élevés, et la correction est encore fréquemment utilisée comme première étape dans de nombreux projets d'analyse de *reads* longs. Toutefois, cette étape peut désormais être efficacement réalisée à l'aide de méthodes d'auto-correction.

En effet, comme l'ont montré les expériences réalisées Section 5.4.2 et Section 5.4.3, les méthodes d'auto-correction tendent à produire des résultats insatisfaisants lorsque les *reads* affichent des taux d'erreurs très importants, mais parviennent à fournir des corrections relativement efficaces lorsque les taux d'erreurs des *reads* atteignent moins de 20%. De ce fait, les méthodes d'auto-correction se sont plus largement développées récemment, profitant non seulement de la réduction des taux d'erreurs des *reads* longs, mais également de la plus grande accessibilité et du coût réduit des technologies de séquençage de troisième génération. En effet, comme le montre la Table 3.2, autant de méthodes d'auto-correction ont été développées uniquement au cours de l'année 2017 qu'en trois ans, entre 2013 et 2015.

De plus, l'évolution des technologies de séquençage de troisième génération n'est pas uniquement marquée par la diminution des taux d'erreurs des *reads* et par les faibles coûts de séquençage. En effet, ces technologies produisent également des débits de *reads* sans cesse croissants, et ces données deviennent donc de plus en plus largement accessibles, soulignant ainsi davantage l'intérêt actuel des méthodes d'auto-correction.

Enfin, la longueur des *reads* est également en constante évolution. Cette évolution est particulièrement marquée par l'apparition récente des librairies *ultra-long reads* ONT, capables de produire des *reads* atteignant jusqu'à un million de paires de bases [46]. Des *reads* atteignant de telles longueurs, bien qu'extrêmement utiles pour la résolution de problèmes d'assemblage, soulèvent cependant de nouvelles problématiques, les méthodes de correction devant en effet être capable de s'y adapter de manière efficace.

6.1.2 Travaux précédents

Comme présenté dans le Chapitre 3, décrivant l'état des méthodes de correction de *reads* longs, l'auto-correction se divise en deux principales approches : l'alignement multiple et l'utilisation de graphes de de Bruijn.

LoRMA, la seule méthode de l'état de l'art permettant de corriger les *reads* longs uniquement à l'aide d'un graphe de de Bruijn, sans étape de calcul de chevauchements, est cependant peu efficace, car nécessitant des profondeurs de séquençage très importantes. En effet, les graphes de de Bruijn successifs utilisés par LoRMA sont construits uniquement à partir des *reads* longs. Ainsi, déterminer les k-mers génomiques et les k-mers contenant des erreurs, uniquement en fonction de leurs fréquences, est difficile. Une importante profondeur de séquençage est alors nécessaire afin d'effectuer une discrimination correcte.

De ce fait, l'ensemble des méthodes d'auto-correction les plus efficaces de l'état de l'art actuel partage une première étape commune de calcul de chevauchements entre les *reads* longs. Ces chevauchements peuvent être calculés à l'aide de méthodes d'alignement, fournissant donc les positions des régions similaires entre les *reads*, mais décrivant également les correspondances base à base entre ces régions, ou à

l'aide de méthode de *mapping*, fournissant uniquement les positions des régions similaires.

Cependant, les méthodes reposant sur le calcul des alignements entre les *reads* longs souffrent d'une consommation de mémoire extrêmement importante, comme l'ont souligné les résultats de Daccord lors des expériences réalisées au sein du Chapitre 5, résumées Table 5.4 et Table 5.5. De plus, ces expériences ont également souligné la difficulté des aligneurs actuels à passer efficacement à l'échelle sur de larges génomes, dû à ces importantes consommations de mémoire. En effet, DALIGNER, l'outil d'alignement utilisé au sein de Daccord, s'est notamment révélé incapable de calculer les alignements entre les *reads* longs d'un jeu de données du génome de *C. elegans*, disposant d'une profondeur de séquençage de 20x, en consommant moins de 32 Go de RAM.

Ainsi, les méthodes reposant sur le calcul des chevauchements entre les *reads* longs via une approche par *mapping* constituent le cœur de l'état de l'art actuel. Les méthodes adoptant une telle approche ne calculent alors les alignements qu'entre les régions chevauchantes mises en évidence par le *mapping*, permettant ainsi de diminuer grandement la comsommation de temps et de mémoire. Une fois l'ensemble des régions d'un *read* considérées, les alignements calculés sont alors généralement résumés à l'aide d'un DAG. Ce DAG est ensuite utilisé afin d'extraire une séquence consensus, permettant de corriger le *reads* initial, en recherchant le chemin de poids maximal.

Cependant, bien que qualifiées de méthodes de correction par alignement multiple, les méthodes reposant sur cette stratégie de résumé des alignements à l'aide d'un DAG ne réalisent en réalité que des *pseudo alignements multiples* entre les *reads*. En effet, le DAG résumant les alignements d'un *read* donné est construit uniquement à partir des alignements deux à deux entre les régions de ce *read*, et les régions des autres *reads* avec lesquelles elles partagent des chevauchements. En particulier, une région d'un *read* donné partageant des chevauchements avec plusieurs autres régions de *reads* différents est alignée indépendamment avec chacune de ces régions chevauchantes. Un alignement indépendant est ainsi généré pour chaque chevauchement, plutôt que de générer un alignement multiple unique à partir de l'ensemble des régions chevauchantes. Bien que plus coûteuse, une telle stratégie d'alignement multiple pourrait cependant impacter positivement les résultats, en faisant usage davantage de séquences, et en construisant ainsi des alignements multiples et des consensus plus précis.

De plus, l'ensemble des méthodes d'auto-correction actuelles, LoRMA exclus, semblent être limitées par la longueur des *reads* à corriger. En particulier, aucune des méthodes de l'état de l'art ne permet de réaliser la correction d'un jeu de données du génome humain comportant des *ultra-long reads* ONT, et ces *reads* doivent donc être filtrés du jeu de données afin de permettre sa correction. Au vu du développement et de l'accessibilité de ces librairies *ultra-long reads*, cette incapacité de passage à l'échelle souligne ainsi une limitation majeure de l'état de l'art actuel.

6.1.3 Contribution

Afin de répondre aux problématiques sus-citées, plus particulièrement concernant le calcul de réels alignements multiples et le passage à l'échelle sur les *ultra-long reads*, nous avons développé CONSENT [1], une nouvelle méthode d'auto-correction. Cette méthode combine différentes approches de l'état de l'art, notamment le calcul

1. https://github.com/morispi/CONSENT

des chevauchements entre les *reads* via une approche par *mapping*, et l'utilisation de graphes de de Bruijn locaux, à la manière de Daccord. Les *reads* sont ainsi corrigés en deux phases, une première par alignement multiple et utilisation de DAG, et une seconde par utilisation de graphes de de Bruijn locaux.

Contrairement aux autres méthodes d'auto-correction par pseudo alignement multiple, CONSENT calcule donc de réels alignements multiples entre les différentes régions chevauchantes des *reads*. Pour cela, afin de réduire le coût de l'alignement multiple, et permettre le passage à l'échelle, aussi bien à d'importants volumes de données qu'aux *ultra-long reads*, la stratégie de segmentation introduite pour ELECTOR, et présentée Section 4.3.2.2, est ici généralisée à un ensemble de taille indéterminée de séquences.

De plus, comme pour ELECTOR, les alignements multiples sont encore une fois calculés à l'aide de POA. Ainsi, le DAG résumant l'ensemble des alignements d'une région donnée d'un *read* peut directement être construit lors de l'étape d'alignement. Une seconde étape distincte de construction du graphe à partir des alignements n'est donc pas nécessaire. Une fois l'alignement multiple calculé et le graphe obtenu, une séquence consensus est alors extraite pour la région du *read*, à la manière des autres méthodes reposant sur une telle stratégie.

Une fois une région donnée d'un *read* long corrigée via cette approche par alignement multiple, l'ensemble des régions des autres *reads* chevauchant cette région est utilisé afin de définir un graphe de de Bruijn local, à la manière de Daccord. Ce graphe est alors utilisé afin de raffiner la correction de la région du *read*, et d'en éliminer les erreurs résiduelles.

D'autre part, CONSENT permet également de corriger des contigs issus d'assemblages de *reads* longs. Pour cela, les chevauchements sont calculés entre les *reads* longs bruts ayant servi à générer l'assemblage et les contigs, et le processus de correction en deux phases décrit précédemment est alors appliqué à partir de ces chevauchements. Bien que l'adaptation à ce problème soit relativement immédiate, CONSENT est la première méthode d'auto-correction permettant réaliser un tel traitement, en plus de la correction classique des *reads*.

6.2 Chaîne de traitement

6.2.1 Vue d'ensemble

Comme mentionné Section 6.1.3, CONSENT combine différentes approches de l'état de l'art. Ainsi, pour commencer, comme la plupart des méthodes d'auto-correction efficaces, CONSENT calcule les chevauchements entre les *reads* en utilisant une approche par *mapping*. Cette étape de calcul des chevauchements est réalisée à l'aide d'un programme externe, et non par CONSENT lui-même. Comme indiqué Section 6.1.2, cette approche de calcul des chevauchements par *mapping* permet, par la suite, de calculer les alignements uniquement entre les régions chevauchantes des *reads*, et donc de diminuer la consommation de ressources.

À la manière de Daccord, la correction d'un *read* est réalisée en divisant celui-ci en un ensemble de fenêtres. Chaque fenêtre est alors corrigée à l'aide des régions des autres *reads* avec lesquelles elle partage des chevauchements. Ainsi, pour une fenêtre donnée d'un *read*, un alignement multiple est calculé à partir de cette fenêtre et de l'ensemble des régions des autres *reads* chevauchant cette fenêtre. Comme pour ELECTOR, ces alignements multiples sont calculés à l'aide de POA, et reposent donc sur des graphes d'alignement d'ordre partiel, tel que décrit Section 4.3.2.1. Ainsi, le DAG représentant l'alignement multiple entre une fenêtre donnée et l'ensemble de

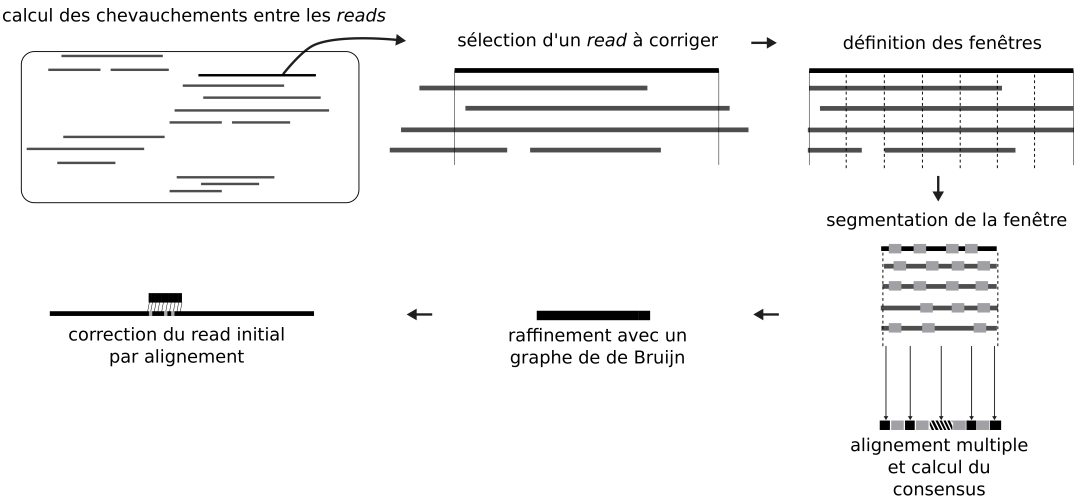

FIGURE 6.1 – **Pipeline de CONSENT.** Les chevauchements entre les *reads* sont calculés via une approche par *mapping*, par défaut, à l'aide de Minimap2. La correction d'un *read* s'effectue alors à partir de ce *read* et de l'ensemble des autres *reads* avec lesquels des chevauchements sont partagés. Le *read* à corriger est divisé en fenêtres, et chaque fenêtre est corrigée indépendamment, à l'aide des séquences des *reads* la chevauchant. Un alignement multiple est alors calculé à partir de l'ensemble de ces séquences, en utilisant une stratégie de segmentation généralisant l'approche introduite pour ELECTOR, décrite Section 4.3.2.2. La séquence consensus de la fenêtre est alors calculée à partir de l'alignement multiple. Cette séquence consensus est ensuite raffinée à l'aide d'un graphe de de Bruijn local, construit à partir de la fenêtre initiale et des séquences la chevauchant. Enfin, la séquence consensus raffinée est réalignée sur le *read* long original, afin de corriger ce dernier. Les autres fenêtres du *read* sont alors traitées de la même manière.

ses séquences chevauchantes est directement construit lors de l'étape d'alignement, et ne nécessite pas de seconde étape de construction distincte, contrairement aux autres méthodes de l'état de l'art. Le graphe peut donc directement être utilisé pour calculer la séquence consensus de la fenêtre, après l'étape d'alignement multiple.

Cependant, cette étape d'alignement multiple, bien qu'appliquée à de petites fenêtres des *reads*, demeure coûteuse en ressources, notamment dû à la dissimilarité entre les séquences induite par les taux d'erreurs des *reads*. Ainsi, afin de réduire le temps nécessaire au calcul des alignements multiples, et donc garantir une bonne capacité de passage à l'échelle, la stratégie de segmentation développée pour ELECTOR, et présentée Section 4.3.2.2, a été généralisée pour CONSENT.

Une fois le consensus d'une fenêtre calculé, une seconde étape de correction est ensuite appliquée. Pour cela, l'ensemble des séquences considérées lors de l'étape d'alignement multiple précédente est utilisé afin de construire un graphe de de Bruijn local, à la manière de Daccord. Ce graphe est alors utilisé afin de raffiner le consensus calculé pour la fenêtre, en corrigeant notamment les régions contenant des k-mers peu fréquents au sein de l'ensemble des séquences utilisé pour construire le graphe. Le taux d'erreurs final du consensus peut ainsi être réduit, afin d'augmenter davantage la qualité de la correction.

Enfin, une fois les deux phases de correction appliquées sur une fenêtre donnée d'un *read*, le consensus raffiné de cette fenêtre est réaligné sur le *read* original, afin de corriger ce dernier. La correction est alors réalisée en traitant indépendamment chacune des fenêtres du *read* de cette façon. Les différentes étapes constituant le pipeline de CONSENT sont illustrées Figure 6.1.

6.2.2 Définitions

Avant de présenter en détail les différentes étapes du pipeline de CONSENT, nous introduisons ici les notions de piles et de fenêtres de chevauchements qui seront utilisées dans le reste de ce chapitre. Ces notions sont extrêmement similaires aux notions de piles et de fenêtres d'alignements introduites par Daccord, bien que légèrement altérées pour les besoins de CONSENT. Ces différences s'expliquent en particulier par la méthodologie utilisée pour calculer les chevauchements entre les *reads*, Daccord utilisant une approche par alignement, et CONSENT une approche par *mapping*.

6.2.2.1 Piles de chevauchements

Une pile de chevauchements représente un ensemble de *reads* partageant des chevauchements avec un *read* A donné. De manière plus formelle, une pile de chevauchements pour un *read* A est définie comme un ensemble de tuples de chevauchement (A_b, A_e, B_b, B_e, S), où :

- A et B représentent les identifiants des *reads* partageant un chevauchement.
- A_b et A_e représentent respectivement les positions de début et de fin du chevauchement sur A.
- B_b et B_e représentent respectivement les positions de début et de fin du chevauchement sur B.
- $S \in \{0, 1\}$ représente l'orientation de B par rapport à A au sein du chevauchement (complémentaire inverse ($S = 1$) ou non ($S = 0$)).

Comme mentionné précédemment, cette définition des piles de chevauchements est extrêmement similaire à la définition des piles d'alignements proposée par Daccord, mais varie légèrement. En particulier, Daccord ajoute un script d'édition au sein des tuples d'alignement, représentant la séquence des opérations d'édition nécessaires afin de transformer $A[A_b..A_e]$ en $B[B_b..B_e]$, si $S = 0$, ou en $\overline{B}[B_b..B_e]$, si $S = 1$ (où $\overline{B}$ représente la séquence complémentaire inverse de B). Ce script d'édition peut facilement être retrouvé par Daccord, sa stratégie de calcul des chevauchements reposant sur une approche par alignement, fournissant donc les correspondances base à base entre les *reads*. Cependant, comme CONSENT repose sur une approche par *mapping* pour calculer les chevauchements, une telle information n'est pas accessible. Les scripts d'édition sont donc exclus de la définition des tuples et des piles de chevauchements.

Au sein de sa pile de chevauchements, le *read* A est qualifié de *read modèle*. La pile de chevauchements d'un *read* modèle A donné contient alors toutes les informations nécessaires à la correction de ce *read*. Une illustration d'une pile de chevauchements est donnée Figure 6.2.

6.2.2.2 Fenêtres sur des piles de chevauchements

En plus du principe des piles d'alignements, Daccord a également souligné l'intérêt de traiter des fenêtres de ces piles, plutôt que de les traiter dans leur intégralité. Ces fenêtres ont également été adaptées aux piles de chevauchements utilisées par CONSENT. Une fenêtre d'une pile de chevauchements représente ainsi un facteur de cette pile. Plus formellement, à partir d'une pile de chevauchements pour un *read* modèle A, une fenêtre de cette pile est définie comme un couple (F_b, F_e), tel que :

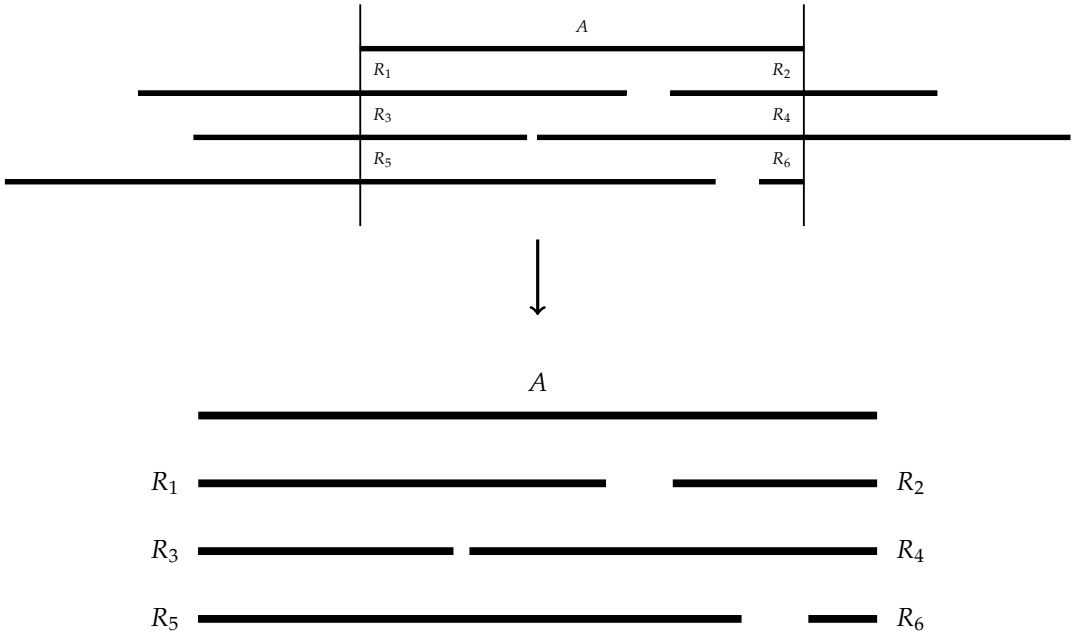

FIGURE 6.2 – **Pile de chevauchements pour un *read* modèle A.** Partie haute : la pile est délimitée par les lignes verticales situées aux extrémités de A. Les préfixes et les suffixes des autres *reads* chevauchant A en dehors de la pile ne sont alors pas considérés lors des étapes suivantes, puisque les informations qu'ils contiennent ne seront pas utiles à la correction de A. Partie basse : la pile de chevauchements de A peut ainsi être représentée en excluant les séquences des *reads* s'étendant en dehors des extrémités de A.

- F_b et F_e représentent respectivement la position de début et la position de fin de la fenêtre, sur A.
- $0 \leq F_b \leq F_e < |A|$, *i.e.* la position de début et la position de fin de la fenêtre définissent un facteur de A.

Ce facteur du *read* A est alors qualifié de *modèle de la fenêtre*. De plus, au sein du pipeline de CONSENT, les deux propriétés suivantes sont ajoutées à la définition des fenêtres, et restreignent donc les fenêtres considérées lors des étapes suivantes :

1. $F_e - F_b + 1 = L$ (*i.e.* les fenêtres ont une longueur fixée).
2. $\forall i, F_b \leq i \leq F_e$, la $i^{\text{ème}}$ base de A est couverte par au moins C *reads* de la pile, A inclus (*i.e.* les fenêtres ont un seuil minimal de couverture).

Cette seconde propriété permet de s'assurer que CONSENT dispose de suffisamment d'information pour calculer un consensus fiable pour chaque fenêtre traitée. Ainsi, CONSENT préfère ne pas traiter certaines fenêtres, si celles-ci ne sont pas suffisamment couvertes. Un tel comportement permet d'éviter d'introduire des biais dans la correction, pouvant être induits par des alignements multiples et des consensus calculés à partir d'un trop faible nombre de séquences. Des exemples de fenêtres, considérées ou non par CONSENT, sont illustrés Figure 6.3.

Dans le cas de Daccord, cette stratégie de division des piles en fenêtres permet de construire des graphes de de Bruijn locaux avec de faibles valeurs de k, afin de contourner les importants taux d'erreurs des *reads* longs, causant des difficultés lors de l'utilisation de graphes de de Bruijn d'ordres trop élevés.

De façon plus générale, traiter des fenêtres plutôt que l'intégralité d'une pile permet de diviser le problème de la correction en de multiples sous-problèmes de plus petite taille, pouvant alors être résolus plus rapidement. En particulier, dans

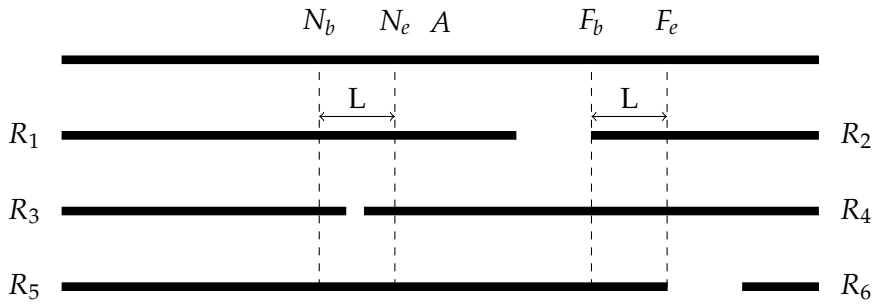

FIGURE 6.3 – **Fenêtres sur une pile de chevauchements.** En fixant la longueur à L et le seuil minimal de couverture à $C = 4$, la fenêtre (F_b, F_e) sera traitée par CONSENT. Avec les mêmes paramètres, la fenêtre (N_b, N_e) ne sera pas traitée par CONSENT. En effet, les bases de A comprises entre les positions N_b et N_e ne sont pas pas toutes couvertes par au moins 4 *reads*, comme le souligne le trou de couverture présent au milieu de la fenêtre.

le cas de CONSENT, la première étape de correction étant réalisée en calculant des alignements multiples, travailler sur des fenêtres permet de diminuer le temps de traitement et la consommation de mémoire, les séquences à aligner étant alors sensiblement plus courtes que les *reads* longs eux-mêmes. De plus, la seconde étape de correction de CONSENT, reposant sur des graphes de de Bruijn, bénéficie ainsi également des avantages liés à l'utilisation de graphes de de Bruijn locaux, mentionnés précédemment.

6.2.3 Calcul des chevauchements

Comme mentionné précédemment, afin de réduire le temps d'exécution et la consommation de mémoire, CONSENT commence par calculer les chevauchements entre les *reads* en utilisant une approche par *mapping*. Par défaut, cette étape est réalisée à l'aide de Minimap2. Cependant, les étapes suivantes, réalisant la correction des *reads* à proprement parler, ne sont pas dépendantes de Minimap2, et d'autres méthodes peuvent ainsi aisément être utilisées pour calculer les chevauchements. La seule restriction est alors que le fichier décrivant les chevauchements entre les *reads* doit être fourni à CONSENT au format PAF. Minimap2 a cependant été choisi comme outil par défaut au vu de ses performances satisfaisantes.

En plus du calcul des chevauchements, lors de cette étape, le fichier décrivant les chevauchements est également indexé à l'aide de fpa [84]. Cet outil a été développé par Pierre Marijon, et permet d'enregistrer, pour chaque *read*, la liste des décalages au sein du fichier PAF, correspondant aux lignes décrivant les chevauchements de ce *read*. En effet, dans le fichier PAF produit par Minimap2, l'ensemble des chevauchements d'un *read* donné ne sont pas toujours consécutifs. Minimap2 peut effectivement construire plusieurs index indépendants à partir de différents sous-ensembles de *reads*, lors du calcul des chevauchements. Un même *read* peut donc partager des chevauchements avec des *reads* situés dans différents index, et ses chevauchements peuvent donc être décrits dans différentes zones du fichier PAF. L'index de fpa permet ainsi de récupérer l'intégralité des chevauchements d'un *read* donné, en naviguant au sein du fichier PAF grâce aux décalages indiqués. De ce fait, il n'est pas nécessaire de parcourir intégralement le fichier PAF afin de retrouver tous les chevauchements d'un *read* donné, ni de le trier afin de regrouper l'ensemble des chevauchements de ce *read* au sein d'une même zone. Éviter ces étapes de tri ou de parcours coûteuses est particulièrement intéressant lors du traitement de larges jeux de données, pouvant résulter en des fichiers PAF de taille importante. De plus,

cet index est stocké sous la forme d'un fichier texte, et ne nécessite pas d'être chargé en mémoire lors de l'exécution.

6.2.4 Calcul des piles de chevauchements et des fenêtres

Les piles de chevauchements sont ainsi calculées à l'aide du fichier PAF et de l'index, produits lors de l'étape précédente. Chaque ligne de l'index représente en effet l'identifiant d'un *read*, ainsi que l'ensemble des décalages, au sein du fichier PAF, correspondant aux lignes décrivant un chevauchement pour ce *read*. Chaque ligne du fichier PAF, décrivant un chevauchement, contient alors l'ensemble des informations nécessaires à la définition d'un tuple de chevauchement : les identifiants des deux *reads* se chevauchant, les positions de début et de fin de ce chevauchement sur les deux *reads*, ainsi que l'orientation du second *read* par rapport au premier. Ainsi, récupérer l'ensemble des lignes du fichier PAF correspondant aux décalages indiqués pour un *read* A de l'index, permet de définir l'ensemble des tuples de chevauchement de ce *read*, et donc sa pile de chevauchements.

À partir de la pile de chevauchements d'un *read* A, il est alors possible de calculer l'ensemble des fenêtres de cette pile. Pour cela, un tableau de la longueur de A est utilisé, afin d'enregistrer le nombre de *reads* de la pile couvrant chaque base de A. Ce tableau est tout d'abord initialisé avec des 1 à chaque position (chaque base de A étant, en effet, au moins couverte par A). Ensuite, pour chaque tuple de chevauchement (A_b, A_e, B_b, B_e, S) inclus dans la pile, les valeurs situées aux positions i, $A_b \leq i \leq A_e$ du tableau sont incrémentées, afin de mettre à jour les valeurs de support de chaque base de A en fonction des différents chevauchements.

Une fois tous les tuples de chevauchement de la pile ainsi considérés, les positions des fenêtres sont retrouvées en recherchant, au sein du tableau, les suites de longueur L de valeurs $\geq C$. En effet, comme mentionné Section 6.2.2.2, CONSENT ne considère pour la correction que des fenêtres de longueur fixée, et respectant un seuil minimal de couverture. Cependant, en pratique, un ensemble de fenêtres chevauchantes est extrait de la pile, plutôt que de partitionner cette dernière en un ensemble de fenêtres non chevauchantes. Ce choix est notamment motivé par le fait que les alignements situés aux extrémités des séquences sont généralement plus difficiles à exploiter que les alignements situés en milieu de séquence. En particulier, ces difficultés peuvent résulter en un manque de séquence consensus aux extrémités de certaines fenêtres. De tels événements causeraient alors un manque de correction au sein du *read*, lors de l'utilisation de fenêtres non chevauchantes.

Une fois l'ensemble des fenêtres d'une pile de chevauchements extrait, chacune de ces fenêtres est alors traitée indépendamment lors des étapes suivantes.

6.2.5 Calcul du consensus d'une fenêtre

Le traitement d'une fenêtre est réalisé en deux étapes distinctes. Tout d'abord, un alignement multiple est calculé à partir de l'ensemble des séquences contenues dans la fenêtre, et une séquence consensus est extraite de cet alignement multiple. Comme pour ELECTOR, cet alignement multiple est calculé à l'aide de graphes d'alignement d'ordre partiel. De plus, la stratégie de segmentation développée pour ELECTOR, et présentée Section 4.3.2.2, est ici généralisée, afin de diviser le problème en plusieurs instances de plus petite taille, et ainsi réduire le temps d'exécution et la consommation de mémoire. Ensuite, une fois le consensus d'une fenêtre calculé, celui-ci est raffiné à l'aide d'un graphe de de Bruijn local, à l'échelle de la fenêtre, afin de corriger les quelques erreurs résiduelles, pouvant encore être présentes.

6.2.5.1 Calcul du consensus par alignement multiple

Comme décrit pour ELECTOR, Section 4.3.2.1, l'alignement multiple des séquences d'une fenêtre est calculé à l'aide de POA. Ainsi, le DAG représentant l'alignement multiple est tout d'abord initialisé avec la séquence du modèle de la fenêtre. Les autres séquences sont ensuite itérativement alignées sur le DAG, celui-ci contenant donc, à chaque étape, le résultat de l'alignement multiple courant. Cependant, contrairement à ELECTOR, calculant des alignements globaux entre les trois versions d'un même *read*, CONSENT calcule plutôt des alignements locaux, afin de considérer uniquement les facteurs des séquences partageant de hauts degrés de similarité avec l'alignement multiple. Ainsi, plutôt que d'utiliser une généralisation de l'algorithme de Needleman-Wunsch, comme dans ELECTOR, les séquences sont ici alignées sur le graphe en utilisant une généralisation de l'algorithme de Smith-Waterman.

Contrairement aux autres méthodes de l'état de l'art, calculant d'abord les alignements deux à deux entre le *read* à corriger et les *reads* le chevauchant, et construisant ensuite un DAG permettant de représenter l'ensemble de ces alignements, cette stratégie permet à CONSENT de directement construire le DAG durant l'étape de calcul de l'alignement multiple. En effet, comme mentionné précédemment, le DAG est tout d'abord initialisé avec la séquence du modèle de la fenêtre, et est ensuite itérativement mis à jour, en y alignant les autres séquences incluses dans la fenêtre, jusqu'à devenir le DAG représentant l'alignement multiple final, une fois toutes ces séquences considérées.

La matrice représentant l'alignement multiple est alors extraite du graphe, et le consensus de la fenêtre est calculé à l'aide d'un vote majoritaire à chacune des positions de cette matrice. Dans les cas où aucune base ne se démarque à une position donnée, la base présente dans la ligne de l'alignement multiple correspondant au modèle de la fenêtre est alors choisie comme consensus à cette position.

6.2.5.2 Généralisation de la stratégie de segmentation

Calculer de tels alignements multiples, même sur des fenêtres de quelques centaines de paires de bases, reste cependant coûteux, notamment lorsque le nombre de séquences à aligner est élevé, ou que la divergence entre les séquences est importante, comme cela peut être le cas avec des *reads* longs. De ce fait, afin de réduire le temps d'exécution de cette étape, la stratégie développée pour ELECTOR, présentée Section 4.3.2.2, et permettant de segmenter l'alignement multiple d'un triplet de séquences, a été généralisée à un ensemble de taille indéterminée de séquences, pour CONSENT.

Ainsi, comme pour ELECTOR, un ensemble de graines est calculé à partir des k-mers apparaissant dans un nombre suffisant de séquences, n'étant répétés dans aucune des séquences, et ne se chevauchant dans aucune des séquences. La contrainte d'ELECTOR, nécessitant qu'un k-mer apparaisse dans l'ensemble des séquences, afin de pouvoir le sélectionner comme graine, est cependant relâchée. En effet, au vu du nombre de séquences à inclure dans les alignements multiples, des taux d'erreurs de ces séquences, et dû au fait que certaines séquences peuvent provenir de régions différentes du génome, à cause de calculs de chevauchements erronés, une telle contrainte ne permettrait pas de définir un nombre de graines suffisant, et ne permettrait donc pas une segmentation satisfaisante.

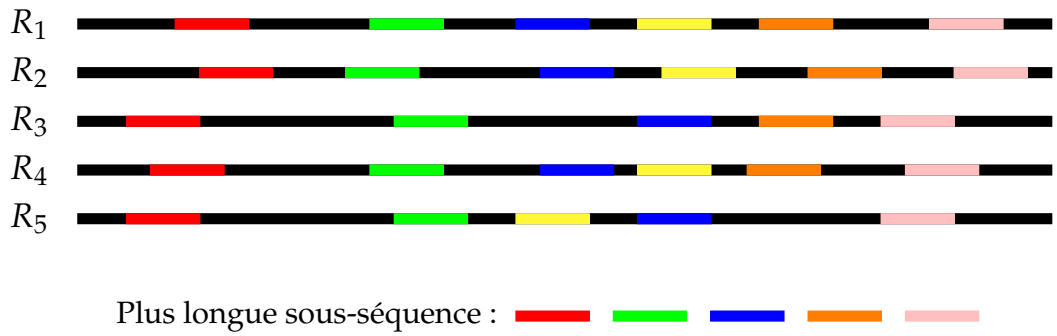

FIGURE 6.4 – **Calcul de la plus longue sous-séquence de graines d'un ensemble de séquences.** Cette sous-séquence est ici calculée en fixant la valeur de la première propriété à $T = 3$, afin de simplifier l'illustration. La plus longue sous-séquence calculée ici est donc $S =$ (rouge, vert, bleu, orange, rose). En ne considérant que les quatre premières séquences, la sous-séquence $S' =$ (rouge, vert, bleu, jaune, orange, rose), de longueur supérieure à S, pourrait être sélectionnée. En effet, bien que R_3 ne contienne pas la graine jaune, la sous-séquence S' respecte les deux propriétés au sein de R_1, R_2, et R_4. Cependant, au sein de R_5, la graine jaune apparaît avant la graine bleue, et invalide donc la seconde propriété. S' ne peut donc pas être sélectionnée comme la plus longue-sous séquence. S est donc choisie. De plus, R_5, ne contenant pas l'ensemble des graines constituant la plus longue sous-séquence S, sera alors filtrée lors de l'étape de calcul des alignements multiples et des consensus.

Comme pour ELECTOR, ces graines sont alors utilisées afin de segmenter l'alignement multiple et le calcul de consensus. Cependant, du fait de la définition différente des graines, le calcul de la plus longue sous-séquence, utilisée pour la segmentation, est également modifié par rapport à ELECTOR. En effet, la plus longue sous-séquence de graines communes à l'ensemble des séquences ne peut pas être calculée ici, pour les raisons mentionnées précédemment. Ainsi, pour CONSENT, la plus longue sous-séquence de graines $S = g_1, g_2, ..., g_n$, respectant les deux propriétés suivantes, est calculée :

1. $\forall i$ tel que $1 \leq i < n$, au moins T séquences contiennent les graines g_i et g_{i+1} (où T est un paramètre fixé à 8 par défaut).
2. $\forall i, j$ tel que $1 \leq i < j \leq n$, la graine g_i apparaît avant la graine g_j, dans l'ensemble des séquences contenant les deux graines g_i et g_j.

Le calcul de cette plus longue sous-séquence de graines est illustré Figure 6.4.

À partir de cette sous-séquence de graines, le problème de l'alignement multiple et du calcul de consensus peut alors être divisé en plusieurs instances, définies sur de petites fenêtres séparées par ces graines, représentant des régions de *matches* exacts entre les séquences. De cette façon, plusieurs alignements multiples et plusieurs consensus indépendants, et de plus petites tailles, peuvent être calculés pour chacune de ces fenêtres. De la même façon que pour la reconstruction de l'alignement multiple d'ELECTOR, le consensus global peut, ici, être reconstruit par concaténation des consensus de chaque fenêtre, et des graines ayant servi à la définition de ces fenêtres. Cette stratégie de segmentation est illustrée Figure 6.5.

De cette façon, les alignements multiples et les consensus sont calculés sur des séquences plus courtes, réduisant ainsi grandement le temps d'exécution. De plus, cette stratégie, permet également de s'assurer que CONSENT n'utilise que des séquences semblables pour la construction de l'alignement multiple et le calcul du consensus, en excluant les séquences trop dissimilaires lors de l'étape de calcul de la plus longue sons-séquence. Ainsi, en plus de réduire le temps d'exécution, cette

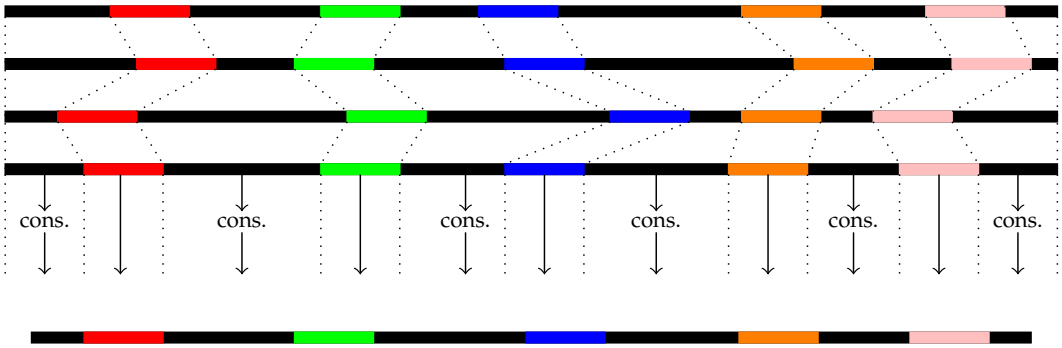

FIGURE 6.5 – **Stratégie de segmentation pour le calcul du consensus d'une fenêtre.** Les graines définissant la plus longue sous-séquence sont utilisées afin de segmenter le calcul de l'alignement multiple et du consensus. Des consensus locaux et indépendants sont ainsi calculés sur les fenêtres des séquences délimitées par ces graines. Les consensus de ces fenêtres, ainsi que les graines, sont ensuite concaténés, afin d'obtenir un consensus final unique, sur toute la longueur des séquences initiales.

stratégie de segmentation permet également de calculer des consensus de meilleure qualité.

6.2.5.3 Raffinement du consensus avec un graphe de de Bruijn

Après le traitement d'une fenêtre, quelques erreurs peuvent encore être présentes au sein du consensus calculé. Ces erreurs sont particulièrement susceptibles d'apparaître dans les cas où la couverture d'une fenêtre est relativement faible, et qu'un consensus de haute qualité ne peut ainsi pas être produit. Afin de limiter le nombre de ces erreurs, après le calcul du consensus, ce dernier est raffiné à l'aide d'un graphe de de Bruijn local, construit à l'échelle de la fenêtre. Cette stratégie permet à CONSENT d'accroître davantage la qualité du consensus, en corrigeant les k-mers le composant, et étant peu fréquents au sein des séquences de la fenêtre.

Pour cela, un graphe de de Bruijn est construit, à partir des k-mers solides des séquences de la fenêtre. Ce graphe de de Bruijn étant local, et réduit à la taille de la fenêtre, une petite valeur de k peut être utilisée pour sa construction. Cela permet à CONSENT de contourner les difficultés habituellement rencontrées, dû aux importants taux d'erreurs des *reads* longs, lors de l'utilisation de graphes de de Bruijn construits pour de larges valeurs de k. De plus, la localité du graphe garantit également une faible consommation de mémoire. De ce fait, le graphe peut être représenté en indexant les k-mers solides des séquences de la fenêtre à l'aide d'une simple table de hachage de la forme (k-mer, nombre d'occurrences), permettant ainsi un accès en temps constant aux arcs de tout sommet du graphe.

Une fois le graphe construit, CONSENT recherche alors, au sein du consensus, des suites de n ($n = 3$ en pratique) k-mers solides, bordant les régions composées de k-mers faibles. CONSENT tente alors de trouver un chemin, au sein du graphe, permettant de relier un k-mer solide de la région bordante gauche à un k-mer solide de la région bordante droite. Une stratégie similaire à celle utilisée au sein de HG-CoLoR, présentée Section 5.3.5, est employée. Les deux k-mers solides des régions bordantes définissent des ancres sur le graphe, et celui-ci est alors traversé afin de trouver un chemin permettant de relier ces deux ancres. De la même façon que pour les explorations du graphe dans HG-CoLoR, une limite sur le nombre maximal d'embranchements à explorer est définie, afin d'éviter les explorations trop

coûteuses. De plus, aux embranchements, les sommets du graphe sont explorés dans l'ordre décroissant du nombre d'occurrences des k-mers qu'ils représentent.

Si aucun chemin ne peut être trouvé afin de relier une paire d'ancres donnée, CONSENT choisit alors deux nouvelles ancres au sein des régions composées de k-mers solides, et recherche un nouveau chemin entre ces deux ancres. Les paires d'ancres sont ainsi considérées dans l'ordre décroissant des nombres d'occurrences cumulés des k-mers les constituant. Lorsqu'un chemin entre deux ancres est trouvé, la région composée de k-mers faibles, située entre ces deux ancres, est alors remplacée par la séquence dictée par le chemin du graphe. Si aucune des paires d'ancres ne peut être reliée à l'aide du graphe, la région composée de k-mers faibles n'est alors pas modifiée.

De la même façon que pour HG-CoLoR, les régions composées de k-mers faibles situées aux extrémités du consensus ne peuvent pas être corrigées à l'aide de cette stratégie. Ainsi, dans ces cas, le graphe est simplement traversé à partir d'un k-mer solide proche de l'extrémité. À chaque embranchement, l'arc menant vers le sommet représentant le k-mer ayant le plus grand nombre d'occurrences est alors choisi. Le graphe est ainsi traversé jusqu'à ce que la longueur du chemin emprunté atteigne la longueur de la région à corriger, ou qu'aucun arc ne puisse être suivi. De la même façon que précédemment, la région composée de k-mers faibles est alors remplacée par la séquence dictée par le chemin du graphe. Ainsi, dans les cas où le chemin du graphe est plus court que la région à corriger, cette dernière est *trimmée*, et corrigée uniquement sur la longueur correspondant au chemin suivi.

6.2.6 Réalignement du consensus d'une fenêtre sur le *read*

Une fois le consensus d'une fenêtre calculé et raffiné, celui-ci doit alors être réancré sur le *read* modèle de la pile. Pour cela, le consensus est aligné sur le *read* modèle, à l'aide d'une version optimisée de l'algorithme de Smith-Waterman [129].

Afin d'éviter de calculer des alignements coûteux en temps, le consensus est cependant uniquement aligné localement, autour des positions de la fenêtre à laquelle il est associé. De cette façon, pour une fenêtre (F_b, F_e) de la pile de chevauchements d'un *read* A, le consensus de cette fenêtre est uniquement aligné sur le facteur $A[F_b - C..F_e + C]$ du *read* A, où C représente la longueur de chevauchement entre les fenêtres consécutives de la pile ($C = 50$ en pratique). Le facteur aligné du *read* est alors remplacé par le facteur aligné du consensus, afin de réaliser la correction.

Aligner le consensus en dehors des extrémités de la fenêtre initiale de cette façon permet de prendre en compte les profils d'erreurs des *reads* longs. En effet, ces derniers contenant majoritairement des erreurs d'insertions et de délétions, il est probable que le consensus calculé à partir d'une fenêtre soit plus long que la fenêtre initiale, et doive donc être réaligné sur une région plus large du *read* modèle.

Cependant, en utilisant une telle approche, les positions d'alignement des consensus de deux fenêtres consécutives de la pile peuvent se chevaucher. Dans ces cas, il est alors nécessaire de déterminer laquelle des deux séquences chevauchantes doit être utilisée comme correction du *read*. Ainsi, dans les cas où les positions d'alignement du consensus de la $i^{\text{ème}}$ fenêtre chevauchent les positions d'alignement du consensus de la $i + 1^{\text{ème}}$ fenêtre, les séquences chevauchantes de ces deux consensus sont calculées. La séquence contenant le plus grand nombre de k-mers solides (les fréquences des k-mers de chaque séquence étant calculées à partir de la fenêtre dont le consensus provient) est alors choisie comme correction pour le *read*. Si les deux séquences contiennent le même nombre de k-mers solides, la séquence du consensus de la $i + 1^{\text{ème}}$ fenêtre est alors arbitrairement choisie comme correction.

Métrique	Sans segmentation	Avec segmentation
Nombre de bases	214 836 958	215 724 911
Taux d'erreurs	0,3394	0,1834
Délétions	592 740	302 677
Insertions	105 575	57 231
Substitutions	31 078	9 302
Rappel	99,9711	99,9816
Précision	99,6659	99,8196
Reads trimmés ou *splittés*	26 411	23 246
Longueur manquante moyenne	91,0	74,8
Reads étendus	1	0
Longueur moyenne d'extension	33,0	0,0
Reads de mauvaise qualité	0	0
Reads trop courts	0	0
Ratio longueur homopolymers	0,9936	1,0000
Temps	5 h 31 min	7 min
Mémoire (Mo)	750	675

TABLE 6.1 – **Comparaison des deux approches de calcul de consensus.** La première approche utilise la stratégie de segmentation, et l'autre calcule les consensus sur l'intégralité de la longueur des fenêtres des piles de chevauchements. Le jeu de données étudié a été simulé à partir du génome d'*E. coli*, avec SimLoRD, en affectant aux *reads* un taux d'erreurs de 12%, et une profondeur de séquençage de 50x. Les deux approches ont été lancées sur 28 threads.

6.2.7 Sortie

CONSENT reporte les *reads* corrigés en version *trimmée*. Les extrémités des *reads* n'ayant pas pu être corrigées, à cause d'un manque de chevauchements, sont ainsi éliminées des *reads* corrigés. De plus, au sein des *reads* corrigés, les bases correspondant à des *k*-mers solides sont reportées en majuscules, et les bases correspondant à des *k*-mers faibles sont reportées en minuscules. Ainsi, bien que CONSENT fournisse uniquement les *reads* corrigés en version *trimmée*, ces derniers peuvent facilement être *splittés* après le processus de correction, afin de n'en conserver que les régions de haute qualité.

6.3 Résultats

6.3.1 Validation de la stratégie de segmentation

Avant de comparer CONSENT aux méthodes d'auto-correction de l'état de l'art, nous étudions tout d'abord dans quelle mesure les résultats obtenus en appliquant la stratégie de segmentation diffèrent des résultats obtenus en calculant les consensus sur l'intégralité de la longueur des fenêtres. Comme pour ELECTOR, nous attendons que les deux approches fournissent des résultats extrêmement similaires, tout en soulignant un important gain de temps lors de l'utilisation de la stratégie de segmentation.

Cette stratégie étant similaire à celle d'ELECTOR, ayant été validée en détail Section 4.5.1, nous présentons ici une unique expérience, réalisée à partir d'un jeu de données simulées du génome d'*E. coli*. Ce jeu de données a été simulé à l'aide de SimLoRD, en affectant aux *reads* un taux d'erreurs de 12%, et une profondeur de

séquençage de 50x. Les *reads* ont alors été corrigées avec CONSENT, avec et sans utilisation de la stratégie de segmentation. La qualité de la correction a ensuite été évaluée à l'aide du premier module d'ELECTOR.

Les résultats de cette expérience sont décrits Table 6.1, et montrent qu'en plus d'être 47 fois plus rapide que l'approche classique, et de consommer légèrement moins de mémoire, la stratégie de segmentation permet également d'obtenir une correction de meilleure qualité. En effet, la stratégie de segmentation permet non seulement de corriger un plus grand nombre de bases, mais aussi de diviser quasiment par deux le taux d'erreurs des *reads* corrigés, et donc d'atteindre un meilleur rappel et une meilleure précision. De plus, un plus faible nombre de *reads* se retrouvent *trimmés* par l'approche utilisant la stratégie de segmentation, et la longueur manquante moyenne des *reads trimmés* par rapport aux *reads* de référence est également plus faible. La même remarque peut être faite sur le nombre de *reads* étendus par la correction, l'approche utilisant la stratégie de segmentation ne produisant aucun *read* étendu sur ce jeu de données. Enfin, la stratégie de segmentation permet également d'atteindre un ratio parfait pour la longueur des homopolymers dans les *reads* corrigés par rapport à la longueur des homopolymers dans les *reads* de référence. Ces résultats confirment donc également la capacité de la stratégie de segmentation à éliminer des calculs des alignements multiples et des consensus les séquences affichant une dissimilarité trop importante avec la majorité des autres séquences, et l'impact positif de cette filtration sur la correction.

6.3.2 Outils et jeux de données évalués

Nous comparons maintenant CONSENT aux autres outils d'auto-correction de l'état de l'art. Les outils suivants sont inclus dans la comparaison : Canu, Daccord, FLAS, LoRMA et MECAT. Les outils de correction hybride et les autres outils d'auto-correction, plus anciens, sont volontairement exclus, afin comparer CONSENT uniquement à des outils d'auto-correction récents.

Ces différents outils sont comparés sur divers jeux de données, d'espèces de taille et de complexité croissantes (*E. coli*, *S. cerevisiae*, *C. elegans*, *D. melanogaster* et *H. sapiens*), composés aussi bien de *reads* simulés PacBio que de *reads* réels ONT. Les données simulées PacBio ont été générées à l'aide de SimLoRD, à partir des génomes d'*E. coli*, de *S. cerevisiae*, et de *C. elegans*, en affectant aux *reads* un taux d'erreurs de 12%, et en générant deux jeux de données pour chaque organisme, affichant respectivement des profondeurs de séquençage de 30x et de 60x. Les données réelles ONT sont, quant à elles, composées d'un jeu de données de *D. melanogaster*, affichant un taux d'erreurs de 14,55% et une profondeur de séquençage de 63x, et d'un jeu de données du premier chromosome humain, affichant un taux d'erreurs de 17,60%, et une profondeur de séquençage de 29x. De plus, ce jeu de données du génome humain contient également des *ultra-long reads*, atteignant des longueurs de 340 000 paires de bases. Les détails sur les génomes de référence utilisés sont présentés Table 6.2. Les détails sur les jeux de données utilisés sont présentés Table 6.3.

Un *benchmark* plus complet, comparant l'intégralité des outils de correction hybride et d'auto-correction sur l'ensemble de ces jeux de données, est également proposé dans le Chapitre 7, Section 7.3.

Tous les outils de correction dont les résultats sont présentés ici, sur données simulées comme sur données réelles, ont été lancés en utilisant leurs paramètres par défaut ou les paramètres recommandés dans leurs publications respectives. Pour CONSENT, la taille des fenêtres a été fixée à 500 paires de bases, la couverture minimale permettant de considérer une fenêtre pour la correction a été fixée à 4, la taille

Génome	Souche	Séquence de référence	Taille du génome (Mbp)
E. coli	K-12 substr. MG1655	NC_000913	4.6
S. cerevisiae	W303	scf7180000000{084-13}	12,2
C. elegans	Bristol N2	GCA_000002985.3	100
D. melanogaster	BDGP Release 6	ISO1 MT/dm6	144
H. sapiens[1]	GRCh38	NC_000001.11	249

TABLE 6.2 – **Description des génomes de référence utilisés pour les comparaisons entre CONSENT et les outils d'auto-correction de l'état de l'art.**
[1] Seul le chromosome 1 a été utilisé.

Jeu de données	Nombre de *reads*	Longueur moyenne (bp)	Couverture	Taux d'erreurs (%)	Numéro d'accession
***Reads* simulés PacBio**					
E. coli 30x	16 959	8 235	30x	12,29	N/A
E. coli 60x	33 918	8 211	60x	12,28	N/A
S. cerevisiae 30x	45 198	8 216	30x	12,28	N/A
S. cerevisiae 60x	90 397	8 204	60x	12,29	N/A
C. elegans 30x	366 416	8 204	30x	12,28	N/A
C. elegans 60x	732 832	8 220	60x	12,28	N/A
***Reads* réels ONT**					
D. melanogaster	1 327 569	6 828	63x	14,55	SRX3676783
H. sapiens[1]	1 075 867	6 744	29x	17,60	PRJEB23027

TABLE 6.3 – **Description des jeux de données utilisés pour les comparaisons entre CONSENT et les outils d'auto-correction de l'état de l'art.**
[1] Seuls les *reads* du chromosome 1 ont été utilisés.

des k-mers a été fixée à 9 pour les étapes de segmentation et de raffinage avec les graphes de de Bruijn locaux, et le seuil de solidité des k-mers a été fixé à 4. De plus, pour les outils le permettant (*i.e.* FLAS et CONSENT), les versions *trimmées* des *reads* ont été évaluées. Pour les autres outils (*i.e.* Canu, Daccord, LoRMA et MECAT), les versions *splittées* des *reads* ont été évaluées.

Pour les jeux de données simulées, les outils de correction ont été lancés sur une machine disposant de 32 Go de RAM, et avec 16 threads. Pour les jeux de données réelles, de plus grande taille, les outils de correction ont été lancés sur un nœud de calcul disposant de 128 Go de RAM, et avec 28 threads.

6.3.3 Impact de l'étape de raffinement de la correction

Nous étudions ici l'impact de l'étape de raffinement de la correction sur les résultats fournis par CONSENT. Ces expériences de validation ont été réalisées sur les jeux de données simulées avec une profondeur de 30x de la Table 6.3, et sont résumées Table 6.4. Ces expériences montrent que cette étape de raffinement impacte de façon non négligeable la correction, permettant de diviser par 1,17 à 1,76 le taux d'erreurs des *reads* corrigés. De la même façon, cette étape de raffinement permet d'obtenir un meilleur rappel et une meilleure précision, mais n'impacte en revanche que peu les métriques concernant les *reads trimmés*, *splittés* ou étendus. De plus, cette étape n'a que très peu d'impact sur le temps d'exécution et la consommation de mémoire, sur les jeux de données *E. coli* et *S. cerevisiae*. En revanche, sur le jeux de données *C. elegans*, l'impact de cette étape est davantage perceptible. Ce comportement peut être expliqué par le fait que les *reads* corrigés du jeu de données *C. elegans* affichent un taux d'erreurs plus élevés, et que le processus de raffinement doit donc traiter un plus grand nombre de bases non corrigées.

Jeu de données	Métrique	Avec raffinement	Sans raffinement
E. coli	Nombre de bases	129 647 002	129 872 304
	Taux d'erreurs (%)	0.3142	0.5547
	Rappel (%)	99.9430	99.8335
	Précision (%)	99.6906	99.4529
	Reads trimmés ou *splittés*	14 681	14 672
	Longueur manquante moyenne (bp)	86,7	86,6
	Reads étendus	0	1
	Longueur moyenne d'extension (bp)	0	21
	Temps	17 min	15 min
	Mémoire (Mo)	2 390	2 337
S. cerevisiae	Nombre de bases	344 457 307	345 174 236
	Taux d'erreurs (%)	0.4091	0.6309
	Rappel (%)	99.9322	99.8199
	Précision (%)	99.5971	99.3775
	Reads trimmés ou *splittés*	38 872	38 829
	Longueur manquante moyenne (bp)	83,8	83,6
	Reads étendus	5	4
	Longueur moyenne d'extension (bp)	118	129,8
	Temps	46 min	42 min
	Mémoire (Mo)	5 523	5 456
C. elegans	Nombre de bases	2 789 537 501	2 795 463 120
	Taux d'erreurs (%)	0.7902	0.9288
	Rappel (%)	99.8773	99.7713
	Précision (%)	99.2204	99.0826
	Reads trimmés ou *splittés*	309 728	311 446
	Longueur manquante moyenne (bp)	80,8	82,5
	Reads étendus	107	115
	Longueur moyenne d'extension (bp)	83,1	79,2
	Temps	9 h 36 min	7 h 07 min
	Mémoire (Mo)	21 819	21 726

TABLE 6.4 – **Impact de l'étape de raffinement de la correction, sur les jeux de données simulées avec une profondeur de 30x.**

6.3.4 Comparaison à l'état de l'art sur données simulées

Les différents outils d'auto-correction mentionnés précédemment ont donc tout d'abord été évalués sur les jeux de données simulées PacBio. Les statistiques décrivant la qualité de la correction fournie par chacun de ces outils ont alors été obtenues en utilisant le premier module d'ELECTOR. Ces résultats sont présentés Table 6.5 et Table 6.6. Seules les principales statistiques reportées par ELECTOR sont présentées ici. L'ensemble des statistiques reportées par ELECTOR est présenté dans le *benchmark* proposé Section 7.3.1.

Le temps d'exécution et la consommation de mémoire de chacun des outils, sur les différents jeux de données, sont présentés Table 6.7. Pour les méthodes ayant des étapes distinctes et aisément identifiables pour le calcul des chevauchements entre les *reads*, et pour la correction à proprement parler (*i.e.* Daccord, MECAT, et CONSENT), le temps d'exécution et la consommation de mémoire de chacune de ces étapes sont également reportés indépendamment.

Sur les jeux de données *E. coli* et *S. cerevisiae*, Daccord a fourni les corrections de meilleure qualité, produisant plus de bases corrigées que l'ensemble des autres outils, et affichant des taux d'erreurs plus faibles, atteignant au plus 0,1259%. Les valeurs reportées pour le rappel, atteignant au moins 99,9874%, se sont également révélées plus élevées que celles de l'ensemble des autres méthodes. De la même façon, les valeurs reportées pour la précision, atteignant au moins 99,8762%, se sont révélées plus élevées que celles de l'ensemble des autres méthodes. De plus, Daccord n'a produit qu'une très faible proportion de *reads* corrigés *trimmés* ou *splittés* par rapport à l'ensemble des autres méthodes. Cependant, l'étape de calcul des chevauchements, reposant sur une approche par alignement, et réalisée à l'aide de DALIGNER, a consommé de très importantes quantités de mémoire, 3 à 11 fois plus élevées que les stratégies de *mapping* employées par CONSENT et par MECAT. Ainsi, Daccord n'est pas parvenu à passer à l'échelle sur les jeux de données *C. elegans*, DALIGNER s'arrêtant en reportant une erreur dès son lancement, même sur le nœud de calcul disposant de 128 Go de RAM.

Au contraire de Daccord, la correction fournie par Canu a affiché des taux d'erreurs bien plus élevés, atteignant au moins 0,4156%, et jusqu'à 1,1052%. Le rappel et la précision reportés pour Canu, atteignant, respectivement, au plus 99,7657% et au plus 99,5887%, se sont également révélés peu satisfaisants comparés aux autres méthodes. Cependant, la consommation de mémoire de Canu s'est révélée faible et stable, grâce à sa stratégie de calcul des chevauchements par *mapping*. En particulier, sur les jeux de données *C. elegans*, Canu s'est révélé extrêmement efficace par rapport aux autres méthodes, ne consommant que 7 Go de RAM.

Sur les jeux de données disposant d'une profondeur de séquençage de 30x, la correction fournie par LoRMA s'est montrée particulièrement insatisfaisante, produisant notamment 4 à 11 fois moins de bases corrigées que l'ensemble des autres méthodes. Les taux d'erreurs, variant de 1,1962 à 3,6960%, ainsi que la précision, chutant jusqu'à 96,3449%, soulignent également la faible qualité des *reads* corrigés par LoRMA. Cette correction de mauvaise qualité peut cependant être expliquée par le fait que LoRMA nécessite d'importantes profondeurs de séquençage pour réaliser une correction efficace, comme mentionné Section 6.1.2. Cette observation est en effet confirmée par les résultats produits par LoRMA sur les jeux de données disposant d'une profondeur de séquençage de 60x, le nombre de bases corrigées demeurant plus faible, mais du même ordre de grandeur que les autres méthodes, sauf pour le jeux de données *C. elegans*. Les taux d'erreurs, variant de 0,1285 à 0,6446%, ainsi que la précision, atteignant au moins 99,3649%, soulignent également une nette

amélioration de la qualité de la correction. Cependant, même sur les jeux de données disposant d'une profondeur de séquençage de 60x, LoRMA a produit une plus grande proportion de *reads trimmés* ou *splittés* que l'ensemble des autres outils, sur tous les jeux de données. De plus, sur l'ensemble des jeux de données, LoRMA a affiché d'importants temps d'exécution, et surtout une consommation de mémoire stable, mais très élevée, atteignant les 32 Go disponibles sur la machine.

MECAT s'est montré extrêmement efficace en termes de temps d'exécution, surpassant l'ensemble des autres méthodes sur tous les jeux de données. Sa stratégie de calcul des chevauchements s'est également révélée hautement efficace, affichant des consommations de mémoire inférieures aux stratégies employées par Daccord et par CONSENT, sur tous les jeux de données. Cependant, la stratégie de *mapping* de MECAT s'est montée plus lente que Minimap2 (l'outil de *mapping* utilisé au sein de CONSENT), la différence de temps entre les deux méthodes s'accentuant en fonction de la taille des jeux de données et de la complexité des génomes. Minimap2 a cependant affiché des consommations de mémoire plus importantes, en particulier sur les jeux de données *C. elegans*.

Cette importante consommation de mémoire de Minimap2 peut être expliquée par le fait que celui-ci ait été lancé en utilisant ses paramètres par défaut lors des expériences réalisées ici. En effet, par défaut, Minimap2 charge en mémoire les *reads*, afin d'en construire un index permettant de réaliser le *mapping*, par paquets de 4 milliards de nucléotides. Une consommation de mémoire plus faible pourrait donc aisément être atteinte en modifiant ce paramètre, et en limitant ainsi la construction des index à des ensembles plus réduits de nucléotides.

En termes de qualité de la correction, MECAT a produit moins de bases corrigées que l'ensemble des autres outils, LoRMA exclus, sur tous les jeux de données. Malgré ce faible débit, les corrections proposées par MECAT se sont révélées satisfaisantes, atteignant des taux d'erreurs d'au plus 0,3908%, et des rappels et des précisions atteignant, respectivement, au moins 99,8903% et au moins 99,6212%. Ces observations, et plus particulièrement le faible nombre de bases corrigées produites, soulignent cependant le fait que MECAT tend à corriger uniquement les *reads* disposant déjà d'un taux d'erreurs relativement faible.

FLAS a permis de corriger un plus grand nombre de bases que MECAT sur l'ensemble des jeux de données. Ceci peut facilement être expliqué par le fait que FLAS est une surcouche de MECAT, permettant de retrouver des chevauchements additionnels, et donc de corriger un plus grand nombre de *reads*. Ainsi, puisque reposant sur la même stratégie de correction, FLAS a affiché des consommations de mémoire hautement similaires à celles de MECAT sur l'ensemble des jeux de données. Les temps d'exécution de FLAS se sont cependant révélés plus élevés que ceux de MECAT, dû aux étapes supplémentaires permettant de retrouver les chevauchements additionnels, et aux plus grands nombres de *reads* ayant ainsi pu être corrigés. Les deux méthodes adoptant un processus de correction extrêmement similaire, la qualité de la correction fournie par FLAS s'est donc montrée comparable à celle fournie par MECAT, atteignant des taux d'erreurs, des rappels, et des précisions similaires, bien que légèrement moins satisfaisants, sur les jeux de données *E. coli* et *S. cerevisiae*. Sur les jeux de données *C. elegans*, et plus particulièrement sur celui disposant d'une profondeur de séquençage de 30x, l'écart de qualité entre les deux méthodes s'est cependant montré plus important, FLAS atteignant notamment un taux d'erreurs de 0,7613%, presque 2 fois plus élevé que celui atteint par MECAT (0,3908%). En revanche, sur l'ensemble des jeux de données, la correction des *reads* par FLAS a permis de grandement réduire le nombre de *reads trimmés*, ainsi que la longueur

manquante au sein de ces *reads*, par rapport à la correction par MECAT, notamment grâce aux alignements additionnels exploités par FLAS.

Les nombres de bases corrigées par FLAS et par CONSENT se sont montrés extrêmement similaires sur l'ensemble des jeux de données, bien que CONSENT affiche un léger avantage sur les jeux de données *C. elegans*. De la même façon, le rappel atteint par les deux outils s'est montré comparable, et même globalement plus élevé pour CONSENT, FLAS ne prenant un léger avantage que sur le jeu de données *C. elegans* disposant d'une profondeur de séquençage de 60x. Au contraire, la précision atteinte par FLAS s'est cependant révélée plus élevée que celle atteinte par CONSENT sur l'ensemble des jeux de données. Ces observations indiquent donc que CONSENT parvient à traiter un plus grand nombre d'erreurs que FLAS, mais que ce dernier corrige convenablement une plus grande proportion des bases identifiées comme incorrectes. Ainsi, bien que comparables, les taux d'erreurs affichés par les deux méthodes donnent un léger avantage à FLAS. Le plus grand écart de qualité peut être observé sur le jeu de données *C. elegans* disposant d'une profondeur de séquençage de 60x. Sur ce jeu de données, FLAS a en effet atteint un taux d'erreurs de 0,3997%, tandis que CONSENT a atteint un taux d'erreurs de 0,5730%, soit seulement 0,1733% plus élevé. Les temps d'exécution des deux outils se sont également montrés comparables sur les jeux de données *E. coli* et *S. cerevisiae*. Cependant, sur les jeux de données *C. elegans*, CONSENT a atteint des temps d'exécution environ 3 fois plus élevés que FLAS.

En revanche, la consommation de mémoire de l'étape de correction de CONSENT s'est montrée plus faible que la consommation de mémoire des étapes de correction de Daccord et de MECAT, sur tous les jeux de données. En particulier, cette étape de correction n'a consommé que légèrement plus de 3 Go de RAM sur le jeu de données *C. elegans* disposant d'une profondeur de séquençage de 60x. De plus, bien que CONSENT produise la plus haute proportion de *reads trimmés* ou *splittés*, après LoRMA, la longueur manquante moyenne au sein de ces *reads* s'est révélée plus faible que la longueur manquante moyenne des *reads trimmés* par l'ensemble des autres méthodes, sur tous les jeux de données, excepté Daccord sur le jeu de données *E. coli* disposant d'une profondeur de séquençage de 60x. La proportion de *reads* étendus après correction par CONSENT s'est également révélée faible comparée aux autres méthodes, LoRMA et MECAT exclus.

6.3.5 Comparaison à l'état de l'art sur données réelles

Les différents outils de correction mentionnés précédemment ont ensuite été évalués sur les jeux de données réelles ONT. Pour ces jeux de données, la qualité des *reads* corrigés ainsi que la qualité des assemblages pouvant être générés à partir de ces *reads* corrigés sont évaluées. Comme pour les expériences sur données réelles réalisées dans le Chapitre 5, le second module d'ELECTOR est utilisé ici, afin de permettre d'évaluer la qualité des *reads* corrigés et des assemblages pouvant être générés.

6.3.5.1 Qualité des *reads* longs corrigés

Les statistiques reportées par le second module d'ELECTOR, pour la partie décrivant la qualité des *reads* corrigés, sont présentées Table 6.8. Le temps d'exécution et la consommation de mémoire de chacun des outils, sur les différents jeux de données, sont présentés Table 6.9. Comme pour les expériences sur données simulées présentées Section 6.3.4, le temps d'exécution et la consommation de mémoire des

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
Canu	Nombre de bases	129 533 580	226 459 133	2 773 456 245
	Taux d'erreurs (%)	0,4156	1,1052	0,5008
	Rappel (%)	99,7647	99,1766	99,7103
	Précision (%)	99,5887	98,9036	99,5040
	Reads trimmés ou *splittés*	9 422	33 164	170 416
	Longueur manquante moyenne (bp)	128,9	1 762,6	126,5
	Reads étendus	55	51	1 100
	Longueur moyenne d'extension (bp)	26,7	28,5	28,1
Daccord	Nombre de bases	131 020 333	347 992 121	-
	Taux d'erreurs (%)	0,0248	0,1259	-
	Rappel (%)	99,9965	99,9874	-
	Précision (%)	99,9757	99,8762	-
	Reads trimmés ou *splittés*	5	102	-
	Longueur manquante moyenne (bp)	478,8	188,5	-
	Reads étendus	0	5	-
	Longueur moyenne d'extension (bp)	0,0	42,4	-
FLAS	Nombre de bases	129 692 582	344 252 752	2 728 691 582
	Taux d'erreurs (%)	0,2720	0,3272	0,7613
	Rappel (%)	99,9291	99,9131	99,8613
	Précision (%)	99,7385	99,6843	99,2541
	Reads trimmés ou *splittés*	1 900	5 082	72 527
	Longueur manquante moyenne (bp)	434,0	438,3	581,6
	Reads étendus	0	1	578
	Longueur moyenne d'extension (bp)	0,0	130,0	161,2
LoRMA	Nombre de bases	2 378 146	14 071 752	32 644 556
	Taux d'erreurs (%)	1,1962	2,1640	3,6960
	Rappel (%)	99,9209	99,8351	99,7269
	Précision (%)	98,8208	97,8564	96,3449
	Reads trimmés ou *splittés*	11 737	35 483	227 760
	Longueur manquante moyenne (bp)	225,6	179,1	158,1
	Reads étendus	0	3	16
	Longueur moyenne d'extension (bp)	0,0	2 204,0	492,7
MECAT	Nombre de bases	106 842 493	284 618 017	2 083 992 906
	Taux d'erreurs (%)	0,2569	0,3040	0,3908
	Rappel (%)	99,9302	99,9160	99,8903
	Précision (%)	99,7533	99,7072	99,6212
	Reads trimmés ou *splittés*	6 625	16 957	148 957
	Longueur manquante moyenne (bp)	1 261,0	1 229,9	1 468,4
	Reads étendus	0	0	9
	Longueur moyenne d'extension (bp)	0,0	0,0	598,1
CONSENT	Nombre de bases	129 647 002	344 457 307	2 789 537 501
	Taux d'erreurs (%)	0,3142	0,4091	0,7902
	Rappel (%)	99,9430	99,9321	99,8773
	Précision (%)	99,6906	99,5971	99,2204
	Reads trimmés ou *splittés*	14 681	38 872	309 728
	Longueur manquante moyenne (bp)	86,7	83,8	80,8
	Reads étendus	0	5	107
	Longueur moyenne d'extension (bp)	0,0	118,0	83,1

TABLE 6.5 – **Comparaison de la qualité de la correction fournie par CONSENT et par les outils d'auto-correction de l'état de l'art, sur des *reads* simulés PacBio, avec une profondeur de séquençage de 30x.** Daccord n'a pas pu corriger le jeu de données *C. elegans*, l'étape d'alignement avec DALIGNER reportant une erreur dès le lancement.

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
Canu	Nombre de bases	218 652 792	599 325 043	5 112 430 638
	Taux d'erreurs (%)	0,7404	0,7919	0,7934
	Rappel (%)	99,4781	99,4488	99,4573
	Précision (%)	99,2658	99,2148	99,2131
	Reads trimmés ou *splittés*	27 690	73 001	569 082
	Longueur manquante moyenne (bp)	985,8	858,1	651,3
	Reads étendus	47	176	1 658
	Longueur moyenne d'extension (bp)	29,0	53,7	37,2
Daccord	Nombre de bases	261 274 775	694 845 174	-
	Taux d'erreurs (%)	0,0214	0,0400	-
	Rappel (%)	99,9971	99,9928	-
	Précision (%)	99,9790	99,9606	-
	Reads trimmés ou *splittés*	13	73	-
	Longueur manquante moyenne (bp)	58,8	276,8	-
	Reads étendus	0	4	-
	Longueur moyenne d'extension (bp)	0,0	3 396,8	-
FLAS	Nombre de bases	260 324 873	689 491 882	5 583 925 852
	Taux d'erreurs (%)	0,1547	0,2034	0,3997
	Rappel (%)	99,9546	99,9418	99,9175
	Précision (%)	99,8526	99,8049	99,6104
	Reads trimmés ou *splittés*	403	1 274	30 950
	Longueur manquante moyenne (bp)	182,4	223,4	305,4
	Reads étendus	1	5	588
	Longueur moyenne d'extension (bp)	102,0	2 655,4	156,7
LoRMA	Nombre de bases	170 581 660	442 516 834	780 732 399
	Taux d'erreurs (%)	0,1285	0,2225	0,6446
	Rappel (%)	99,9865	99,9785	99,9547
	Précision (%)	99,8743	99,7812	99,3649
	Reads trimmés ou *splittés*	40 347	108 516	1 154 643
	Longueur manquante moyenne (bp)	1 007,0	970,3	412,3
	Reads étendus	0	4	13
	Longueur moyenne d'extension (bp)	0,0	663,0	685,7
MECAT	Nombre de bases	232 519 767	616 247 287	4 937 676 257
	Taux d'erreurs (%)	0,1714	0,2088	0,2675
	Rappel (%)	99,9547	99,9428	99,9258
	Précision (%)	99,8362	99,7996	99,7415
	Reads trimmés ou *splittés*	3 622	9 606	112 978
	Longueur manquante moyenne (bp)	365,0	380,7	566,8
	Reads étendus	0	0	30
	Longueur moyenne d'extension (bp)	0,0	0,0	165,0
CONSENT	Nombre de bases	259 078 481	689 502 352	5 603 760 432
	Taux d'erreurs (%)	0,1722	0,2918	0,5730
	Rappel (%)	99,9843	99,9481	99,9119
	Précision (%)	99,8306	99,7125	99,4345
	Reads trimmés ou *splittés*	27 380	71 027	57 1107
	Longueur manquante moyenne (bp)	71,8	68,1	67,1
	Reads étendus	0	2	187
	Longueur moyenne d'extension (bp)	0,0	854,5	122,2

TABLE 6.6 – **Comparaison de la qualité de la correction fournie par CONSENT et par les outils d'auto-correction de l'état de l'art, sur des *reads* simulés PacBio, avec une profondeur de séquençage de 60x.** Daccord n'a pas pu corriger le jeu de données *C. elegans*, l'étape d'alignement avec DALIGNER reportant une erreur dès le lancement.

Jeu de données	Outil	Chevauchements		Correction		Total	
		Temps	Mémoire (Mo)	Temps	Mémoire (Mo)	Temps	Mémoire (Mo)
E. coli 30x	Canu	-	-	-	-	19 min	4 613
	Daccord	1 min	6 813	13 min	639	14 min	6 813
	FLAS	-	-	-	-	12 min	1 639
	LoRMA	-	-	-	-	10 min	32 155
	MECAT	25 sec	1 600	1 min	1 083	2 min	1 600
	CONSENT	22 sec	2 390	17 min	132	17 min	2 390
E. coli 60x	Canu	-	-	-	-	24 min	3 674
	Daccord	3 min	18 450	51 min	1 191	54 min	18 450
	FLAS	-	-	-	-	38 min	2 428
	LoRMA	-	-	-	-	1 h 39 min	31 682
	MECAT	1 min	2 387	4 min	1 553	5 min	2 387
	CONSENT	1 min	4 849	35 min	248	36 min	4 849
S. cerevisiae 30x	Canu	-	-	-	-	29 min	3 681
	Daccord	7 min	31 798	1 h 12 min	3 487	1 h 19 min	31 798
	FLAS	-	-	-	-	29 min	2 935
	LoRMA	-	-	-	-	46 min	31 899
	MECAT	1 min	2 907	4 min	1 612	5 min	2 907
	CONSENT	1 min	5 523	45 min	284	46 min	5 523
S. cerevisiae 60x	Canu	-	-	-	-	1 h 11 min	3 710
	Daccord	10 min	32 190	2 h 16 min	1 160	2 h 26 min	32 190
	FLAS	-	-	-	-	1 h 30 min	4 984
	LoRMA	-	-	-	-	5 h 25 min	31 828
	MECAT	4 min	4 954	12 min	2 412	16 min	4 954
	CONSENT	2 min	11 325	1 h 44 min	535	1 h 46 min	11 325
C. elegans 30x	Canu	-	-	-	-	9 h 09 min	6 921
	Daccord	-	-	-	-	-	-
	FLAS	-	-	-	-	3 h 07 min	10 565
	LoRMA	-	-	-	-	8 h 19 min	31 827
	MECAT	27 min	10 535	21 min	2 603	48 min	10 535
	CONSENT	15 min	21 819	9 h 21 min	1 885	9 h 36 min	21 819
C. elegans 60x	Canu	-	-	-	-	9 h 30 min	7 050
	Daccord	-	-	-	-	-	-
	FLAS	-	-	-	-	10 h 45 min	13 682
	LoRMA	-	-	-	-	31 h 04 min	32 104
	MECAT	1 h 28 min	10 563	1 h 15 min	3 775	2 h 43 min	10 563
	CONSENT	57 min	32 284	26 h 07 min	3 390	27 h 04 min	32 284

TABLE 6.7 – **Comparaison du temps d'exécution et de la consommation de mémoire de CONSENT et des différents outils d'auto-correction de l'état de l'art, sur des *reads* simulés PacBio.** Daccord n'a pas pu corriger les deux jeux de données *C. elegans*, l'étape d'alignement avec DALIGNER reportant une erreur dès le lancement.

étapes de calcul des chevauchements et de correction sont reportées indépendamment pour les méthodes le permettant.

Sur ces deux jeux de données, Daccord n'a pas pu être lancé, DALIGNER ne parvenant pas à réaliser l'étape d'alignement des *reads* et reportant une erreur dès le lancement, comme lors des expériences sur les jeux de données *C. elegans* simulés.

Les *reads* corrigés par LoRMA ont affiché la plus haute identité moyenne, atteignant jusqu'à 98,4749% sur le jeu de données *D. melanogaster*. Cependant, la longueur moyenne des *reads* corrigés s'est montrée extrêmement faible, atteignant au plus 458 bases sur le jeu de données *D. melanogaster*, et rendant ces *reads* corrigés davantage comparables à des *reads* courts qu'à des *reads* longs. Ces observations confirment donc la tendance de LoRMA à *splitter* un grand nombre de *reads*, tel qu'observé lors des expériences sur les données simulées réalisées Section 6.3.4. Cette faible longueur moyenne et cette grande proportion de *reads splittés* peuvent également expliquer les importantes identités observées. En effet ces statistiques soulignent que LoRMA ne parvient pas à corriger les régions complexes et répétées, et se contente alors de corriger de courts fragments des *reads*, pouvant aisément être résolus par l'utilisation des graphes de de Bruijn. Ainsi, LoRMA a également affiché la plus faible proportion de couverture du génome, sur ces deux jeux de données. En particulier, sur le jeu de données du génome humain, les alignements n'ont permis de couvrir que 28,6186% du génome. Pour l'ensemble de ces raisons, LoRMA est exclus du reste de la discussion.

Sur ces deux jeux de données, CONSENT a corrigé le plus grand nombre de *reads*. Ces *reads* corrigés ont également affiché la plus haute identité, atteignant jusqu'à 93,0004% sur le jeu de données *H. sapiens*, contre seulement 91,6925% et 90,9952% pour les *reads* corrigés respectivement par MECAT et par FLAS. CONSENT a également produit le plus grand nombre de bases corrigées, et atteint la plus importante proportion de couverture du génome, pour ces deux jeux de données.

Sur le jeu de données *D. melanogaster*, le seul ayant pu être corrigé, Canu a produit des résultats globalement comparables aux autres méthodes, bien que moins de *reads*, et donc moins de bases, aient pu êtres corrigés. La longueur moyenne des *reads* corrigés s'est cependant montrée plus importante que toutes les autres méthodes, MECAT exclus. La couverture du génome s'est, quant à elle, révélée plus élevée que MECAT, mais plus faible que FLAS et CONSENT. De la même façon, l'identité moyenne des *reads* corrigés s'est montrée plus élevée que FLAS, mais plus faible que MECAT et CONSENT. De plus, bien que plus efficace en termes de consommation de mémoire, Canu s'est montré plus lent que FLAS et MECAT, mais plus rapide que CONSENT.

En termes de temps d'exécution et de consommation de mémoire, comme lors des expériences sur données simulées, MECAT s'est montré plus efficace que l'ensemble des autres méthodes. De plus, MECAT a atteint la plus importante proportion de *reads* alignés, sur les deux jeux de données. La proportion de *reads* alignés après correction par CONSENT s'est cependant montrée comparable à celle de MECAT, seuls 0,36 à 0,61% de *reads* supplémentaires pouvant être alignés après correction par ce dernier.

Contrairement aux observations réalisées lors des expériences sur données simulées, la correction fournie par CONSENT s'est ici montrée plus satisfaisante que la correction fournie par FLAS. L'ensemble des métriques produites ont en effet souligné un net avantage pour CONSENT, hormis pour la longueur moyenne des *reads* corrigés.

De plus, CONSENT a été le seul outil capable de passer à l'échelle et de corriger les *ultra-longs reads* présents dans le jeu de données *H. sapiens*. En effet, lors du

traitement de ces *reads*, l'ensemble des autres méthodes se sont arrêtées en reportant une erreur, hormis LoRMA, ce dernier produisant cependant des résultats insatisfaisants, comme décrit précédemment. Ainsi, pour tout de même permettre aux autres méthodes de corriger ce jeu de données, les *reads* plus longs que 50 000 paires de bases ont été filtrés. Le jeu de données initial contenait 1 824 de ces *reads*, représentant un total de 135 364 312 paires de bases. Cependant, même après filtration de ces *ultra-long reads*, Canu n'est toujours pas parvenu à corriger le jeu de données, et s'est de nouveau arrêté en reportant une erreur, soulignant ainsi l'importante difficulté de passage à l'échelle de cet outil.

Concernant l'étape de correction, CONSENT s'est à nouveau montré légèrement plus efficace que MECAT en termes de consommation de mémoire, sur le jeu de données *D. melanogaster*. La légère surconsommation de mémoire de CONSENT par rapport à MECAT, sur le jeu de données *H. sapiens*, peut toutefois être expliquée par la présence des *ultra-long reads* au sein du jeu de données corrigé par CONSENT. Cependant, la tendance de CONSENT à afficher des temps d'exécution plus importants que FLAS s'est confirmée ici, en particulier sur le jeu de données *D. melanogaster*, la correction par CONSENT nécessitant 38h, soit presque 4 fois plus de temps que FLAS. En revanche, sur le jeu de données *H. sapiens*, malgré la présence des *ultra-long reads*, le temps d'exécution de CONSENT ne s'est montré que 1,7 fois plus élevé que FLAS.

6.3.5.2 Qualité des assemblages

Les statistiques reportées par le second module d'ELECTOR, pour la partie décrivant la qualité des assemblages générés à partir des *reads* corrigés, sont présentées Table 6.10.

Au vu de la longueur des *reads* produits par LoRMA, aucun assemblage n'a pu être généré pour le jeu de données *D. melanogaster*. Pour le jeu de données *H. sapiens*, bien qu'un assemblage ait pu être généré, ce dernier, composé uniquement de 4 contigs particulièrement courts, n'a permis de couvrir que 0,0060% du génome. Au vu de ces résultats, LoRMA est de nouveau exclus du reste de la discussion.

Sur le jeu de données *D. melanogaster*, les *reads* corrigés par MECAT ont produit l'assemblage le moins satisfaisant. En effet, ce dernier est composé du plus grand nombre de contigs, affiche les tailles NGA50 et NGA75 les plus faibles, et couvre la plus faible proportion du génome de référence. Les *reads* corrigés par FLAS, plus nombreux que ceux corrigés par MECAT, ont permis d'obtenir un assemblage de meilleure qualité, composé de moins de contigs, et affichant une taille NGA50 presque 5 fois plus élevée. De plus, ces contigs ont permis de couvrir une plus large proportion du génome. L'assemblage généré à partir des *reads* corrigés par Canu, bien que composé d'un plus grand nombre de contigs que l'assemblage généré à partir des *reads* corrigés par FLAS, a atteint des tailles NGA50 et NGA75 plus élevées, et a permis de couvrir une proportion du génome similaire, bien que légèrement plus faible. Bien que composé du plus grand nombre de contigs, après l'assemblage généré à partir des *reads* corrigés par MECAT, les *reads* corrigés par CONSENT ont permis de générer l'assemblage affichant les plus grandes tailles NGA50 et NGA75, respectivement 1,91 et 2,42 fois plus élevées que celles de l'assemblage généré à partir des *reads* corrigés par FLAS. De plus, l'assemblage généré à partir des *reads* corrigés par CONSENT a également permis de couvrir la plus large proportion du génome de référence.

Sur le jeu de données *H. sapiens*, la tendance précédemment observée entre MECAT et FLAS s'est inversée. En effet, malgré le plus grand nombre de *reads* corrigés

Outil	Métrique	Jeu de données	
		D. melanogaster	*H. sapiens*
Canu[1]	Nombre de *reads*	829 965	-
	Nombre de bases	6 992 565 713	-
	Longueur moyenne (bp)	8 425	-
	Reads alignés (%)	98,0543	-
	Identité moyenne (%)	95,2043	-
	Couverture du génome (%)	97,8902	-
Daccord[1]	Nombre de *reads*	-	-
	Nombre de bases	-	-
	Longueur moyenne (bp)	-	-
	Reads alignés (%)	-	-
	Identité moyenne (%)	-	-
	Couverture du génome (%)	-	-
FLAS[1]	Nombre de *reads*	938 392	717 973
	Nombre de bases	7 866 243 928	5 695 206 077
	Longueur moyenne (bp)	8 382	7 932
	Reads alignés (%)	95,6517	99,0611
	Identité moyenne (%)	94,9925	90,9952
	Couverture du génome (%)	98,1012	92,3690
LoRMA	Nombre de *reads*	13 920 047	6 796 439
	Nombre de bases	6 386 351 401	1 247 317 428
	Longueur moyenne (bp)	458	183
	Reads alignés (%)	97,0463	96,4997
	Identité moyenne (%)	98,4749	97,8254
	Couverture du génome (%)	94,7561	28,6189
MECAT[1]	Nombre de *reads*	849 704	667 532
	Nombre de bases	7 287 782 016	5 479 325 177
	Longueur moyenne (bp)	8 576	8 208
	Reads alignés (%)	99,8681	99,9495
	Identité moyenne (%)	96,5229	91,6925
	Couverture du génome (%)	97,3432	91,4387
CONSENT	Nombre de *reads*	1 065 621	869 462
	Nombre de bases	8 178 293 508	6 348 740 755
	Longueur moyenne (bp)	7 674	7 301
	Reads alignés (%)	99,2593	99,5864
	Identité moyenne (%)	96,7179	93,0004
	Couverture du génome (%)	98,1981	92,3993

TABLE 6.8 – **Comparaison de la qualité des *reads* longs corrigés par CONSENT et par les outils de correction de l'état de l'art, sur des *reads* réels ONT.** Daccord n'a pas pu corriger ces jeux de données, l'étape d'alignement avec DALIGNER reportant une erreur dès le lancement. Canu s'est également arrêté en reportant une erreur lors de la correction du jeu de données *H. sapiens*, malgré la filtration des *ultra-long reads*.

[1] Les *reads* plus longs que 50 000 paires de bases ont été filtrés du jeu de données *H. sapiens*, car les programmes s'arrêtaient en reportant une erreur lors de leur traitement. Le jeu de données initial contenait 1 824 de ces *ultra-long reads*, représentant un total de 135 364 312 paires de bases.

Jeu de données	Outil	Chevauchements		Correction		Total	
		Temps	Mémoire (Mo)	Temps	Mémoire (Mo)	Temps	Mémoire (Mo)
D. melanogaster	Canu	_	_	_	_	14 h 04 min	10 295
	Daccord	_	_	_	_	_	_
	FLAS	_	_	_	_	10 h 18 min	18 820
	LoRMA	_	_	_	_	23 h 51 min	65 536
	MECAT	28 min	13 443	1 h 26 min	7 724	1 h 54 min	13 443
	CONSENT	43 min	51 361	37 h 17 min	7 570	38 h 00 min	51 361
H. sapiens	Canu[1]	_	_	_	_	_	_
	Daccord[1]	_	_	_	_	_	_
	FLAS[1]	_	_	_	_	4 h 57 min	14 957
	LoRMA	_	_	_	_	13 h 07 min	50 435
	MECAT[1]	26 min	11 075	1 h 27 min	4 591	1 h 53 min	11 075
	CONSENT	17 min	45 869	8 h 13 min	5 759	8 h 30 min	45 869

TABLE 6.9 – **Comparaison du temps d'exécution et de la consommation de mémoire de CONSENT et des différents outils d'auto-correction de l'état de l'art, sur des *reads* réels ONT.** Daccord n'a pas pu corriger ces jeux de données, l'étape d'alignement avec DALIGNER reportant une erreur dès le lancement. Canu s'est également arrêté en reportant une erreur lors de la correction du jeu de données *H. sapiens*, malgré la filtration des *ultra-long reads*.
[1] Les *reads* plus longs que 50 000 paires de bases ont été filtrés du jeu de données *H. sapiens*, car les programmes s'arrêtaient en reportant une erreur lors de leur traitement.

par FLAS, ces derniers ont généré un assemblage moins satisfaisant, composé de plus de contigs, affichant des tailles NGA50 et NGA75 plus courtes, et couvrant une proportion du génome de référence légèrement plus faible. Les *reads* corrigés par CONSENT ont permis de générer un assemblage contenant uniquement un contig de moins que l'assemblage généré à partir des *reads* corrigés par MECAT, malgré la présence des *ultra-long reads*. De plus, parmi ces contigs, 3 n'ont pas pu être alignés sur le génome de référence, contre seulement 2 au sein de l'assemblage généré à partir des *reads* corrigés par MECAT. L'assemblage généré à partir des *reads* corrigés par CONSENT a cependant atteint des tailles NGA50 et NGA75 respectivement 2,10 et 2,95 fois plus élevées que l'assemblage généré à partir des *reads* corrigés par MECAT, et a également permis de couvrir une proportion du génome de référence légèrement plus élevée.

Ainsi, ces observations soulignent que, pour ces deux jeux de données, les *reads* corrigés par CONSENT ont permis de générer des assemblages globalement plus satisfaisants que les *reads* corrigés par les autres méthodes.

6.3.6 Correction d'assemblages

Comme mentionné Section 6.1.3, en plus de l'auto-correction, CONSENT dispose également d'une fonctionnalité additionnelle permettant de corriger des assemblages de *reads* longs. L'adaptation de CONSENT à la problématique de la correction d'assemblages est en effet relativement immédiat. Plus précisément, plutôt que de calculer les chevauchements entre les *reads* longs, comme présenté Section 6.2.3, les chevauchements sont simplement calculés entre les contigs et les *reads* longs ayant servi à générer l'assemblage. Le reste du pipeline de CONSENT reste alors identique.

Nous présentons ici des résultats de correction d'assemblages sur les jeux de données simulées *E. coli*, *S. cerevisiae* et *C. elegans* disposant d'une profondeur de séquençage de 60x, ainsi que sur les deux jeux de données réelles *D. melanogaster* et *H. sapiens*. Nous comparons les résultats fournis par CONSENT aux résultats fournis par RACON [111], un outil de l'état de l'art permettant de corriger des assemblages. L'outil QUAST-LG [87] a été utilisé ici, afin d'évaluer la qualité des assemblages, avant et après correction, plus précisément qu'avec le second module

Outil	Métrique	Jeu de données	
		D. melanogaster	*H. sapiens*
Canu[1]	Nombre de contigs	276	-
	Nombre de contigs alignés	269	-
	Nombre de *breakpoints*	1 269	-
	NGA50 (bp)	4 517 869	-
	NGA75 (bp)	1 293 578	-
	Couverture du génome (%)	92,1034	-
Daccord[1]	Nombre de contigs	-	-
	Nombre de contigs alignés	-	-
	Nombre de *breakpoints*	-	
	NGA50 (bp)	-	-
	NGA75 (bp)	-	-
	Couverture du génome (%)	-	-
FLAS[1]	Nombre de contigs	265	259
	Nombre de contigs alignés	264	258
	Nombre de *breakpoints*	1 379	317
	NGA50 (bp)	4 220 587	1 622 624
	NGA75 (bp)	830 179	346 914
	Couverture du génome (%)	92,1105	88,4068
LoRMA	Nombre de contigs	-	4
	Nombre de contigs alignés	-	4
	Nombre de *breakpoints*	-	0
	NGA50 (bp)	-	2 742
	NGA75 (bp)	-	2 742
	Couverture du génome (%)	-	0,0060
MECAT[1]	Nombre de contigs	370	249
	Nombre de contigs alignés	370	248
	Nombre de *breakpoints*	1 058	199
	NGA50 (bp)	853 544	1 783 022
	NGA75 (bp)	278 555	462 658
	Couverture du génome (%)	89,5839	88,7002
CONSENT	Nombre de contigs	321	248
	Nombre de contigs alignés	317	245
	Nombre de *breakpoints*	2 508	871
	NGA50 (bp)	8 077 282	3 760 455
	NGA75 (bp)	2 009 827	1 368 818
	Couverture du génome (%)	92,5688	88,9587

TABLE 6.10 – **Comparaison de la qualité des assemblages générés à partir des *reads* longs corrigés par CONSENT et par les outils d'auto-correction de l'état de l'art, sur des *reads* réels ONT.** Daccord n'a pas pu corriger ces jeux de données, l'étape d'alignement avec DALIGNER reportant une erreur dès le lancement. Canu s'est également arrêté en reportant une erreur lors de la correction du jeu de données *H. sapiens*, malgré la filtration des *ultra-long reads*. Les assemblages ne sont donc pas évalués pour ces deux outils. De plus, les *reads* corrigés par LoRMA n'ont pas permis de produire un assemblage pour le jeu de données *D. melanogaster*.

[1] Les *reads* plus longs que 50 000 paires de bases ont été filtrés du jeu de données *H. sapiens*, car les programmes s'arrêtaient en reportant une erreur lors de leur traitement.

d'ELECTOR. En particulier, QUAST-LG permet, en effet, de fournir des informations sur le nombre d'erreurs contenues dans un assemblage par rapport au génome de référence auquel cet assemblage est comparé. Les résultats de ces expériences sont présentés Table 6.11. Les temps d'exécution et les consommations de mémoire présentés dans cette table décrivent uniquement l'étape de correction, et non l'étape de calcul des chevauchements entre les *reads* et les contigs. CONSENT et RACON se basent en effet tous les deux sur Minimap2 lors de cette étape, et l'inclure au sein de la comparaison ne serait donc pas informatif.

Ces résultats montrent que CONSENT se révèle plus efficace que RACON en termes de qualité des résultats, plus précisément en termes de réduction du taux d'erreurs, sur les jeux de données simulées *E. coli*, *S. cerevisiae* et *C. elegans*. De plus, les tailles NGA50 et NGA75 des assemblages corrigés avec CONSENT, bien que légèrement plus faibles, se sont montrées très similaires à celles des assemblages corrigés avec RACON. L'assemblage corrigé avec RACON a cependant permis de couvrir une proportion du génome de référence légèrement plus importante (0,11%) sur le jeu de données *C. elegans*.

Sur le jeux de données réelles *D. melanogaster*, l'assemblage corrigé par RACON a affiché un plus faible taux d'erreurs et a permis de couvrir une proportion du génome légèrement plus importante (0,66%) que l'assemblage corrigé par CONSENT. Les tailles NGA50 et NGA75 des assemblages corrigés par les deux outils se sont cependant montrées comparables, l'assemblage corrigé par CONSENT affichant même une taille NGA50 plus élevée que l'assemblage corrigé par RACON. De plus, après correction par CONSENT, un contig de plus a pu être aligné sur le génome de référence.

Les différences observées et l'avantage de RACON se sont cependant accentués sur le jeu de données *H. sapiens*. L'assemblage corrigé par RACON a en effet affiché un taux d'erreurs 1,91 fois plus faible, des tailles NGA50 et NGA75 2 fois plus élevées, et a permis de couvrir une proportion du génome de référence 2% plus importante que l'assemblage corrigé par CONSENT.

Cependant, sur tous les jeux de données évalués, CONSENT s'est montré 1,36 à 16,75 fois plus rapide que RACON. La consommation de mémoire de CONSENT, bien plus élevée que celle de RACON sur l'ensemble des jeux de données, peut être expliquée par le fait que CONSENT charge, par défaut, les informations nécessaires à la correction de 1 000 *reads* en mémoire. Ainsi, dans les expériences réalisées ici, les informations nécessaires à la correction de l'intégralité des contigs ont donc été chargées en mémoire en même temps. De plus, les contigs étant plus longs que les *reads*, un plus grand nombre de chevauchements est associé à chacun d'eux, expliquant également l'importante différence de consommation de mémoire par rapport aux expériences réalisées pour la correction de *reads*, présentées Table 6.7 et Table 6.9.

Les différences observées au niveau de la qualité des résultats fournis par RACON et par CONSENT, plus particulièrement sur les jeux de données *D. melanogaster* et *H. sapiens*, peuvent également être expliquées par le fait que CONSENT ait été utilisé sans ajustement des paramètres, ni pour l'étape de calcul des chevauchements, ni pour l'étape de correction. Adapter davantage ces paramètres au problème de la correction d'assemblages pourrait ainsi permettre d'obtenir de meilleurs résultats.

De plus, ces résultats de correction d'assemblages de *reads* bruts se sont montrés comparables aux résultats d'assemblage obtenus après correction des *reads*, présentés Section 6.3.5.2. Sur le jeu de données *H. sapiens*, la correction de l'assemblage des *reads* bruts a même permis d'obtenir moins de contigs, et de couvrir une plus large proportion du génome, au prix d'une réduction des tailles NGA50 et NGA75.

Ces résultats soulignent ainsi l'intérêt des méthodes d'auto-correction a également permettre une adaptation au problème de la correction d'assemblages.

6.4 Synthèse

Dans ce chapitre, nous avons décrit CONSENT, une nouvelle méthode d'auto-correction de *reads* longs combinant différentes approches de l'état de l'art. Les chevauchements entre les *reads* sont tout d'abord calculés à l'aide d'une stratégie de *mapping*, et deux étapes de correction, d'abord par alignement multiple, puis par graphe de de Bruijn, sont ensuite appliquées. Pour cela, CONSENT commence par diviser les régions chevauchantes des *reads* en de plus petites fenêtres, afin de calculer des alignements multiples et des consensus indépendants pour chacune de ces fenêtres. Les alignements multiples sont calculés à l'aide d'une méthode basée sur des graphes d'alignement d'ordre partiel, permettant de produire de réels alignements multiples, contrairement aux autres méthodes de l'état de l'art actuel. De plus, cette méthode d'alignement multiple est combinée à une généralisation de la stratégie de segmentation développée pour ELECTOR, et présentée Section 4.3.2.2, permettant ainsi une importante réduction du temps de calcul des alignements. Une fois le consensus d'une fenêtre extrait de l'alignement multiple, celui-ci est alors raffiné à l'aide d'un graphe de de Bruijn local, afin d'en éliminer les erreurs résiduelles. Enfin, le consensus raffiné est réaligné sur le *read* original, afin de réaliser la correction.

Les expériences réalisées, aussi bien sur des données simulées PacBio que sur des données réelles ONT, montrent que les résultats fournis par CONSENT sont qualitativement comparables aux résultats fournis par les autres méthodes d'auto-correction de l'état de l'art. Sur les données réelles ONT considérées, plus bruitées que les données simulées PacBio, CONSENT a même permis de produire des corrections de meilleure qualité que les autres méthodes. La qualité de la correction fournie par CONSENT a également permis de générer des assemblages de bonne qualité, notamment meilleurs que les assemblages générés à partir des *reads* corrigés par les autres méthodes, en termes de tailles NGA50 et NGA75, ainsi qu'en termes de taux de couverture des génomes de référence.

De plus, ces expériences ont également souligné que CONSENT est actuellement la seule méthode d'auto-correction capable de passer à l'échelle sur un jeu de données du génome humain contenant des *ultra-long reads* ONT, atteignant jusqu'à 340 000 paires de bases. Bien que les librairies permettant de générer ces *ultra-long reads* soient récentes, leur développement et leur accessibilité devraient s'accroître dans un futur proche. Ainsi, passer à l'échelle et pouvoir corriger efficacement de tels *reads* va rapidement devenir une nécessité pour les méthodes d'auto-correction. CONSENT pourrait donc être le premier outil d'auto-correction pouvant être appliqué à de tels *reads*, à une plus grande échelle.

Une fonctionnalité additionnelle de correction d'assemblages a également été ajoutée à CONSENT. En effet, le processus de correction de *reads* et le processus de correction d'assemblages sont algorithmiquement très proches. L'adaptation à ce problème est alors relativement immédiate, seule l'étape de calcul des chevauchements variant d'un problème à l'autre. Malgré cette adaptation aisée, CONSENT est la première méthode d'auto-correction de l'état de l'art permettant de traiter les deux problèmes. De plus, les expériences réalisées soulignent que CONSENT permet de produire des résultats qualitativement comparables aux résultats fournis par RACON, une méthode spécifiquement développée pour la correction d'assemblages,

Jeu de données	Métrique	Outil		
		Aucun	RACON	CONSENT
E. coli 60x	Nombre de contigs	1	1	1
	Nombre de contigs alignés	1	1	1
	NGA50 (bp)	4 939 014	4 663 914	4 637 939
	NGA75 (bp)	4 939 014	4 663 914	4 637 939
	Couverture du génome (%)	99,91	99,91	99,91
	Erreurs / 100 kbp	10 721	499	78
	Temps (secondes CPU)	N/A	5 597	334
	Mémoire (Mo)	N/A	643	1 648
S. cerevisiae 60x	Nombre de contigs	29	29	29
	Nombre de contigs alignés	29	29	29
	NGA50 (bp)	579 247	539 472	532 258
	NGA75 (bp)	456 470	346 116	334 560
	Couverture du génome (%)	96,14	96,09	96,15
	Erreurs / 100 kbp	10 694	637	208
	Temps (secondes CPU)	N/A	14 931	1 616
	Mémoire (Mo)	N/A	1 703	7 073
C. elegans 60x	Nombre de contigs	47	47	47
	Nombre de contigs alignés	47	47	47
	NGA50 (bp)	5 495 235	5 073 456	5 019 450
	NGA75 (bp)	2 656 350	2 349 027	2 286 190
	Couverture du génome (%)	99,71	99,72	99,61
	Erreurs / 100 kbp	10 611	819	330
	Temps (secondes CPU)	N/A	136 325	30 907
	Mémoire (Mo)	N/A	14 288	85 486
D. melanogaster	Nombre de contigs	423	422	422
	Nombre de contigs alignés	408	416	417
	NGA50 (bp)	864 011	693 918	704 027
	NGA75 (bp)	159 590	287 386	250 324
	Couverture du génome (%)	83,19	92,88	92,22
	Erreurs / 100 kbp	10 690	973	2 028
	Temps (secondes CPU)	N/A	197 124	82 006
	Mémoire (Mo)	N/A	19 508	74 446
H. sapiens	Nombre de contigs	201	200	200
	Nombre de contigs alignés	188	191	192
	NGA50 (bp)	1 025 355	1 482 262	761 519
	NGA75 (bp)	-	456 658	226 997
	Couverture du génome (%)	77,75	95,82	93,83
	Erreurs / 100 kbp	11 139	2 337	4 474
	Temps (secondes CPU)	N/A	141 923	104 117
	Mémoire (Mo)	N/A	16 202	71 402

TABLE 6.11 – **Comparaison de la qualité des assemblages bruts, après correction avec RACON, et après correction avec CONSENT.** QUAST-LG n'a pas reporté de métrique pour la taille NGA75 de l'assemblage brut sur le jeu de données *H. sapiens*.

tout en étant jusqu'à 16,75 fois plus rapide. La qualité de la correction fournie par CONSENT tend cependant à diminuer à mesure que la complexité du génome augmente, notamment dû au fait que CONSENT ait été lancé avec ses paramètres par défaut lors de ces expériences. Adapter davantage ces paramètres au problème de la correction d'assemblages pourrait alors permettre d'obtenir de meilleurs résultats.

Ces résultats demeurent néanmoins satisfaisants, et pourraient donc inspirer les autres méthodes d'auto-correction à proposer une telle fonctionnalité de correction d'assemblages, au prix d'une faible charge de travail additionnelle.

Chapitre 7

Benchmark des méthodes de correction de *reads* longs

Sommaire

7.1 Introduction

Dans ce chapitre, nous présentons un *benchmark* complet des méthodes de correction hybride et d'auto-correction décrites au Chapitre 3, et résumées Table 3.1 et Table 3.2. Ces méthodes de correction sont évaluées sur les jeux de données réelles et simulées utilisés au sein du Chapitre 5, décrivant HG-CoLoR, et au sein du Chapitre 6, décrivant CONSENT. Ces jeux de données sont présentés Table 5.3 et Table 6.3. Les évaluations ont été réalisées à l'aide d'ELECTOR, en utilisant le premier module pour les données simulées, et le second module pour les données réelles, afin d'évaluer aussi bien la qualité des *reads* longs corrigés que la qualité des assemblages.

Les méthodes suivantes sont cependant exclues du *benchmark*, pour les raisons mentionnées ci-dessous :

- FalconSense : cet outil n'est parvenu à s'installer sur aucune des machines ayant servi à réaliser les différentes expériences.
- HECIL : cet outil n'est parvenu à corriger aucun *read*, et a produit un ensemble de *reads* corrigés identiques aux *reads* originaux, pour l'ensemble des jeux de données.
- LSCplus : cet outil n'est plus disponible en téléchargement.
- MiRCA : cet outil s'est arrêté en reportant une erreur lors de la correction de l'ensemble des jeux de données.

- PBcR : cet outil s'est arrêté en reportant une erreur lors de la correction de l'ensemble des jeux de données.
- PBcR-BLASR et PBcR-MHAP : ces deux outils sont des versions préliminaires du module de correction de l'assembleur Canu. Seul ce dernier, plus récent, est donc évalué au sein de *benchmark*.
- PBDAGCon : cet outil n'a produit aucun *read* corrigé sur l'ensemble des jeux de données.
- Sparc : cet outil s'est arrêté en reportant une erreur lors de la correction de l'ensemble des jeux de données.
- Sprai : cet outil s'est arrêté en reportant une erreur lors de la correction de l'ensemble de jeux de données.

Dans ces expériences, nous nous intéressons principalement aux avantages et aux inconvénients de la correction hybride et de l'auto-correction, en fonction des propriétés des différents jeux de données étudiés.

7.2 Jeux de données du Chapitre 5

7.2.1 Données simulées

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
CoLoRMap	Nombre de bases	64 353 820	81 060 058	210 966 804	571 054 288
	Taux d'erreurs (%)	0,1336	0,1946	0,2655	2,6255
	Délétions	71 639	165 412	434 983	14 745 290
	Insertions	7 062	9 193	51 627	374 284
	Substitutions	51 954	62 188	279 378	3 722 575
	Rappel (%)	99,9923	99,9890	99,9805	99,8445
	Précision (%)	99,8707	99,8118	99,7413	97,4526
	Reads trimmés ou *splittés*	1 666	2 067	6 970	64 051
	Longueur manquante moyenne (bp)	939,2	1 497,5	1 396,2	3 234,8
	Reads étendus	8 463	10 730	28 117	96 953
	Longueur moyenne d'extension (bp)	260,3	308,8	251,6	233
	Reads de mauvaise qualité	12	20	121	332
	Reads trop courts	689	1 036	2 166	181 741
	Ratio longueur homopolymers	1,0000	1,0000	0,9826	0,9962
	Temps	57 min	1 h 25 min	4 h 42 min	125 h 44 min
	Mémoire (Mo)	6 198	9 659	13 544	32 188
ECTools	Nombre de bases	36 892 324	6 261 214	116 347 153	54 323
	Taux d'erreurs (%)	1,2086	1,3991	1,2428	1,4652
	Délétions	77 464	13 883	246 662	123
	Insertions	395 759	76 702	1 275 965	702
	Substitutions	40 670	7 026	140 703	45
	Rappel (%)	98,8241	98,6228	98,8048	98,5410
	Précision (%)	98,7923	98,6012	98,7581	98,5348
	Reads trimmés ou *splittés*	5 628	1 595	18 270	18
	Longueur manquante moyenne (bp)	2 730,3	4 103,1	2 747,6	4 770,3
	Reads étendus	0	0	0	0
	Longueur moyenne d'extension (bp)	0	0	0	0
	Reads de mauvaise qualité	0	0	0	0
	Reads trop courts	0	0	0	0
	Ratio longueur homopolymers	1,0064	1,0000	0,9991	1,0045
	Temps	17 min	8 min	57 h 12 min	77 h 59 min
	Mémoire (Mo)	862	917	2 256	5 028
FMLRC	Nombre de bases	64 715 552	83 732 334	223 529 739	1 832 773 801
	Taux d'erreurs (%)	0,1194	0,1930	1,0016	3,3582
	Délétions	26 253	54 707	643 464	20 717 465
	Insertions	89 640	185 196	2 289 467	52 872 229
	Substitutions	22 637	45 795	546 972	16 628 231
	Rappel (%)	99,9142	99,8498	99,1797	97,8709
	Précision (%)	99,8815	99,8081	99,0020	96,6643
	Reads trimmés ou *splittés*	20	28	114	3 090
	Longueur manquante moyenne (bp)	26,2	29,4	17,2	32,8
	Reads étendus	0	0	9	295
	Longueur moyenne d'extension (bp)	0	0	290,8	31
	Reads de mauvaise qualité	13	29	120	744
	Reads trop courts	0	0	0	0
	Ratio longueur homopolymers	1,0000	1,0000	0,9741	1,0000
	Temps	23 min	29 min	1 h 19 min	8 h 02 min
	Mémoire (Mo)	387	408	906	7 939

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
HALC	Nombre de bases	63 708 698	81 199 351	212 266 193	1 588 220 052
	Taux d'erreurs (%)	0,1113	0,1537	0,4212	1,5288
	Délétions	59 157	115 973	1 035 978	35 970 722
	Insertions	8 273	15 034	100 874	2 520 363
	Substitutions	5 206	13 764	198 853	2 919 936
	Rappel (%)	99,9937	99,9918	99,9734	99,8945
	Précision (%)	99,8909	99,8497	99,5872	98,4961
	Reads trimmés ou *splittés*	2 358	3 816	12 043	153 855
	Longueur manquante moyenne (bp)	341,7	547,3	577,5	777,2
	Reads étendus	24	24	71	772
	Longueur moyenne d'extension (bp)	25,2	31,1	53,2	40,9
	Reads de mauvaise qualité	2	2	160	2 582
	Reads trop courts	101	254	3 436	170 934
	Ratio longueur homopolymers	1,0000	1,0000	1,0580	0,9978
	Temps	22 min	24 min	1 h 19 min	5 h 59 min
	Mémoire (Mo)	597	965	2 115	2 323
Hercules	Nombre de bases	70 375 149	91 064 066	242 189 693	1 976 944 911
	Taux d'erreurs (%)	10,5079	10,6411	10,9420	11,9249
	Délétions	1 319 811	1 753 865	4 655 235	40 229 132
	Insertions	7 040 655	9 211 444	25 099 150	216 151 231
	Substitutions	1 113 866	1 447 112	4 003 822	35 147 508
	Rappel (%)	89,6376	89,4974	89,3519	88,3131
	Précision (%)	89,4946	89,3613	89,0615	88,0792
	Reads trimmés ou *splittés*	8	14	42	706
	Longueur manquante moyenne (bp)	22,2	20,5	20,1	30,4
	Reads étendus	5	14	65	133
	Longueur moyenne d'extension (bp)	22,6	24,6	194,1	23
	Reads de mauvaise qualité	13	29	143	678
	Reads trop courts	0	0	0	0
	Ratio longueur homopolymers	0,9982	1,0000	1,0000	1,0014
	Temps	6 h 23 min	7 h 47 min	32 h 38 min	199 h 42 min
	Mémoire (Mo)	1 724	1 633	18 771	31 039
HG-CoLoR	Nombre de bases	65 065 102	84 089 814	219 744 436	1 726 223 265
	Taux d'erreurs (%)	0,0430	0,0691	0,2959	0,6524
	Délétions	11 561	28 955	339 174	9 986 160
	Insertions	15 803	33 409	548 419	5 156 404
	Substitutions	1 869	3 147	79 127	775 253
	Rappel (%)	99,9991	99,9982	99,9900	99,9682
	Précision (%)	99,9574	99,9315	99,7071	99,3554
	Reads trimmés ou *splittés*	133	498	4 562	71 079
	Longueur manquante moyenne (bp)	264,7	274,5	677,4	866,3
	Reads étendus	5 969	8 180	19 237	140 791
	Longueur moyenne d'extension (bp)	67	71,1	72,5	72,2
	Reads de mauvaise qualité	2	0	99	621
	Reads trop courts	0	4	496	8 849
	Ratio longueur homopolymers	1,0000	1,0000	0,9741	1,0000
	Temps	51 min	51 min	4 h 55 min	88 h 10 min
	Mémoire (Mo)	1 432	1 517	3 237	19 730

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Jabba	Nombre de bases	68 653 634	87 851 697	216 794 630	1 480 673 578
	Taux d'erreurs (%)	0,0862	0,0636	0,1058	0,1909
	Délétions	53 665	42 166	112 063	1 865 824
	Insertions	9 913	7 041	95 757	649 951
	Substitutions	1 919	466	21 954	102 639
	Rappel (%)	99,9977	99,9975	99,9973	99,9929
	Précision (%)	99,9150	99,9370	99,8950	99,8106
	Reads trimmés ou *splittés*	639	1 204	6 387	87 670
	Longueur manquante moyenne (bp)	3 323,5	2 735,3	2 166,6	2 853,8
	Reads étendus	8 267	10 402	25 214	168 261
	Longueur moyenne d'extension (bp)	850,4	922,7	1 080,5	1 043,1
	Reads de mauvaise qualité	10	18	115	661
	Reads trop courts	179	386	2 954	34 239
	Ratio longueur homopolymers	1,0000	1,0000	0,9563	1,0000
	Temps	2 min	2 min	5 min	45 min
	Mémoire (Mo)	1 217	1 218	1 217	13 360
LoRDEC	Nombre de bases	60 892 408	77 969 503	188 228 237	1 154 508 245
	Taux d'erreurs (%)	0,0712	0,1474	0,5400	1,2643
	Délétions	48 836	128 344	1 385 989	23 642 857
	Insertions	8 473	21 205	122 252	1 584 355
	Substitutions	6 144	17 896	229 925	1 845 645
	Rappel (%)	99,9921	99,9890	99,9483	99,8871
	Précision (%)	99,9313	99,8570	99,4730	98,7542
	Reads trimmés ou *splittés*	2 951	4 786	22 470	210 051
	Longueur manquante moyenne (bp)	1 155,2	1 033,9	857,6	1 350,9
	Reads étendus	0	0	3	30
	Longueur moyenne d'extension (bp)	0	0	1 285,3	156,9
	Reads de mauvaise qualité	2	0	252	1 966
	Reads trop courts	260	737	22 811	474 725
	Ratio longueur homopolymers	1,0000	1,0000	0,9741	0,9995
	Temps	6 min	8 min	28 min	6 h 01 min
	Mémoire (Mo)	438	455	799	2 238
LSC	Nombre de bases	37 111 831	44 310 080	122 224 269	771 778 062
	Taux d'erreurs (%)	5,8260	5,2598	6,1549	6,7716
	Délétions	372 086	438 428	1 630 573	14 922 136
	Insertions	1 835 695	1 991 268	6 295 946	40 404 903
	Substitutions	253 915	286 969	895 587	5 874 295
	Rappel (%)	94,6923	95,2419	94,6340	94,7734
	Précision (%)	94,2025	94,7706	93,8772	93,2623
	Reads trimmés ou *splittés*	5 990	7 307	20 257	147 403
	Longueur manquante moyenne (bp)	3 008,4	3 031,8	2 962,6	2 934,9
	Reads étendus	24	85	145	289
	Longueur moyenne d'extension (bp)	22,7	24	24,3	26,1
	Reads de mauvaise qualité	0	0	47	271
	Reads trop courts	599	944	2 453	29 243
	Ratio longueur homopolymers	1,0026	1,0000	0,9833	0,9938
	Temps	47 min	47 min	9 h 42 min	117 h 34 min
	Mémoire (Mo)	337	328	1 756	1 852

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Nanocorr[1]	Nombre de bases	64 372 749	83 338 435	220 133 609	-
	Taux d'erreurs (%)	0,4099	0,3043	0,4723	-
	Délétions	72 260	77 573	276 444	-
	Insertions	301 605	261 194	1 023 357	-
	Substitutions	54 832	57 152	258 241	-
	Rappel (%)	99,6491	99,7715	99,6679	-
	Précision (%)	99,5928	99,6989	99,5322	-
	Reads trimmés ou *splittés*	1 574	1 602	5 076	-
	Longueur manquante moyenne (bp)	322,8	279,4	349,6	-
	Reads étendus	0	0	1	-
	Longueur moyenne d'extension (bp)	0	0	20	-
	Reads de mauvaise qualité	1	0	46	-
	Reads trop courts	0	0	0	-
	Ratio longueur homopolymers	1,0000	1,0000	1,0158	-
	Temps	2 h 52 min	3 h 17 min	39 h 52 min	-
	Mémoire (Mo)	173	166	2 345	-
NaS[2]	Nombre de bases	58 963 977	78 034 042	207 085 378	-
	Taux d'erreurs (%)	2,1095	1,3035	2,7483	-
	Délétions	714 139	530 425	2 102 677	-
	Insertions	1 100 737	948 398	5 112 655	-
	Substitutions	94 241	86 401	504 562	-
	Rappel (%)	99,9158	99,9468	99,9077	-
	Précision (%)	97,8969	98,7009	97,2615	-
	Reads trimmés ou *splittés*	339	274	2 187	-
	Longueur manquante moyenne (bp)	2 361,4	2 370,8	2 437,7	-
	Reads étendus	6 688	8 070	22 658	-
	Longueur moyenne d'extension (bp)	1 371,8	1 781,2	1 488,4	-
	Reads de mauvaise qualité	1 651	1 751	5 040	-
	Reads trop courts	0	0	0	-
	Ratio longueur homopolymers	1,0000	1,0000	1,0000	-
	Temps	24 h 24 min	28 h 48 min	217 h 20 min	-
	Mémoire (Mo)	2 099	2 099	2 099	-
Proovread	Nombre de bases	62 131 071	80 950 481	210 578 217	1 651 572 671
	Taux d'erreurs (%)	0,1122	0,1220	0,2075	0,5320
	Délétions	73 995	127 118	670 850	21 888 508
	Insertions	1 364	2 255	26 710	239 748
	Substitutions	1 066	3 344	30 808	464 923
	Rappel (%)	99,9944	99,9938	99,9880	99,9599
	Précision (%)	99,8897	99,8808	99,7964	99,4820
	Reads trimmés ou *splittés*	3 221	4 485	14 211	98 127
	Longueur manquante moyenne (bp)	299,5	268,1	332,5	599,5
	Reads étendus	11	18	45	242
	Longueur moyenne d'extension (bp)	27,3	23,8	257	79,3
	Reads de mauvaise qualité	1	0	124	1 357
	Reads trop courts	861	752	2 834	24 984
	Ratio longueur homopolymers	1,0000	1,0000	0,9669	0,9935
	Temps	52 min	1 h 18 min	4 h 24 min	79 h 33 min
	Mémoire (Mo)	5 784	6 029	10 435	20 213

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Canu	Nombre de bases	66 632 351	86 443 218	229 555 492	1 873 188 109
	Taux d'erreurs (%)	4,9496	5,2422	5,0619	4,9561
	Délétions	738 704	1 011 985	2 574 320	20 233 048
	Insertions	3 517 266	4 804 729	12 252 413	97 517 920
	Substitutions	622 602	841 679	2 197 172	17 594 481
	Rappel (%)	95,2430	94,9490	95,1477	95,2663
	Précision (%)	95,0573	94,7647	94,9459	95,0524
	Reads trimmés ou *splittés*	547	764	2 216	14 407
	Longueur manquante moyenne (bp)	27,1	27,2	35,1	29,5
	Reads étendus	53	66	178	1 462
	Longueur moyenne d'extension (bp)	32,3	27,4	30,7	32,5
	Reads de mauvaise qualité	0	0	43	305
	Reads trop courts	0	0	0	0
	Ratio longueur homopolymers	1,0055	1,0000	0,9937	1,0120
	Temps	12 min	14 min	32 min	4 h 37 min
	Mémoire (Mo)	2 974	2 821	3 274	6 950
CONSENT	Nombre de bases	48 390 086	61 487 428	166 101 116	1 359 131 353
	Taux d'erreurs (%)	8,2534	8,5423	8,2652	9,5548
	Délétions	4 778 369	6 371 753	16 549 340	155 680 250
	Insertions	1 613 389	2 076 177	5 404 368	50 056 903
	Substitutions	114 628	142 283	455 785	6 084 369
	Rappel (%)	97,9538	97,9155	98,0349	97,9553
	Précision (%)	91,8579	91,5687	91,8483	90,5794
	Reads trimmés ou *splittés*	8 332	10 835	28 621	230 249
	Longueur manquante moyenne (bp)	1 507,1	1 581,7	1 498	1 401
	Reads étendus	100	112	267	2 373
	Longueur moyenne d'extension (bp)	133,8	110,3	106,9	92,2
	Reads de mauvaise qualité	0	0	57	471
	Reads trop courts	0	0	0	0
	Ratio longueur homopolymers	1,0070	1,0000	0,9788	0,9765
	Temps	6 min	8 min	22 min	3 h 49 min
	Mémoire (Mo)	1 020	1 552	4 514	14 522
Daccord[3]	Nombre de bases	64 669 977	83 773 362	222 050 951	-
	Taux d'erreurs (%)	0,4694	0,3965	0,5447	-
	Délétions	121 756	75 627	470 101	-
	Insertions	275 390	321 532	977 499	-
	Substitutions	25 146	33 860	200 541	-
	Rappel (%)	99,8647	99,8817	99,8591	-
	Précision (%)	99,5371	99,6077	99,4630	-
	Reads trimmés ou *splittés*	328	119	991	-
	Longueur manquante moyenne (bp)	445,5	369,9	410,4	-
	Reads étendus	0	0	0	-
	Longueur moyenne d'extension (bp)	0	0	0	-
	Reads de mauvaise qualité	0	0	49	-
	Reads trop courts	17	4	54	-
	Ratio longueur homopolymers	1,0061	1,0000	0,9810	-
	Temps	16 min 23	24 min	1 h 10 min	-
	Mémoire (Mo)	3 509	4 538	14 111	-

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
FLAS	Nombre de bases	35 567 295	44 048 436	125 103 108	662 199 783
	Taux d'erreurs (%)	2,1958	2,2848	2,4742	9,1804
	Délétions	777 449	879 707	2 963 186	71 270 551
	Insertions	590 885	801 154	2 362 512	40 983 626
	Substitutions	75 331	89 736	322 562	9 060 762
	Rappel (%)	99,6847	99,6914	99,6456	99,2995
	Précision (%)	97,8309	97,7424	97,5575	90,8881
	Reads trimmés ou *splittés*	4 992	6 372	17 066	110 818
	Longueur manquante moyenne (bp)	2 040,3	2 097,1	2 051,1	1 806,9
	Reads étendus	44	78	215	5 956
	Longueur moyenne d'extension (bp)	196,4	343,2	280	198,2
	Reads de mauvaise qualité	1 868	2 591	6 081	85 161
	Reads trop courts	97	149	339	2 725
	Ratio longueur homopolymers	0,9806	1,0000	0,9783	0,9964
	Temps	2 min	3 min	9 min	1 h 13 min
	Mémoire (Mo)	1 231	1 354	2 259	10 371
LoRMA	Nombre de bases	676 120	884 592	6 279 242	11 394 637
	Taux d'erreurs (%)	0,6305	0,9261	1,6624	5,0838
	Délétions	3 032	5 139	195 481	996 348
	Insertions	1 112	744	21 776	83 683
	Substitutions	1 156	2 136	14 201	110 710
	Rappel (%)	99,9766	99,9733	99,8771	99,7216
	Précision (%)	99,3790	99,0871	98,3563	94,9446
	Reads trimmés ou *splittés*	340	667	2 990	36 351
	Longueur manquante moyenne (bp)	3 183,5	3 115,5	1 927,7	1 165,9
	Reads étendus	0	0	6	24
	Longueur moyenne d'extension (bp)	0	0	1 383,2	195,4
	Reads de mauvaise qualité	14	0	201	2 476
	Reads trop courts	446	1 248	8 683	125 774
	Ratio longueur homopolymers	1,0000	1,0000	1,0000	1,0000
	Temps	5 min	5 min	20 min	3 h 42 min
	Mémoire (Mo)	32 118	32 183	32 183	32 041
MECAT	Nombre de bases	46 854 371	58 979 203	162 057 920	870 965 775
	Taux d'erreurs (%)	0,5340	0,5243	0,6555	0,6540
	Délétions	223 296	267 547	975 507	5 379 870
	Insertions	66 725	82 121	263 724	1 566 201
	Substitutions	5 883	7 575	52 975	139 668
	Rappel (%)	99,8289	99,8317	99,8015	99,8196
	Précision (%)	99,4817	99,4915	99,3636	99,3597
	Reads trimmés ou *splittés*	4 802	6 304	16 034	120 325
	Longueur manquante moyenne (bp)	2 036,7	2 114,5	2 069,5	2 320,8
	Reads étendus	0	0	1	41
	Longueur moyenne d'extension (bp)	0	0	23	354,7
	Reads de mauvaise qualité	0	0	50	583
	Reads trop courts	0	0	0	1
	Ratio longueur homopolymers	1,0008	1,0057	0,9810	0,9938
	Temps	22 sec	26 sec	1 min 20 sec	18 min 20 sec
	Mémoire (Mo)	1 202	1 322	2 207	10 340

TABLE 7.1 – **Comparaison de la qualité de la correction fournie par les différentes méthodes de l'état de l'art, sur les jeux de données simulées de la Table 5.3.** [1] Nanocorr n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [2] NaS n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [3] Daccord n'a pas pu corriger le jeu de données *C. elegans*, l'étape d'alignement avec DALIGNER consommant une quantité de mémoire trop importante.

La majorité des méthodes d'auto-correction n'est pas parvenue à corriger efficacement ces jeux de données. En effet, parmi les six outils d'auto-correction évalués, seuls Daccord et MECAT ont permis une correction satisfaisante, réduisant les taux d'erreurs des *reads* en dessous de 1%. Deux limitations peuvent cependant être identifiées pour ces outils. D'une part, par rapport aux outils de correction hybride, et même par rapport à Daccord, Canu et CONSENT, MECAT n'est parvenu à corriger qu'un faible nombre de *reads* pour l'ensemble des jeux de données, comme les soulignent les métriques décrivant le nombre de bases des *reads* corrigés. D'autre part, comme mentionné Section 5.4.2, Daccord n'est pas parvenu à corriger le jeu de données *C. elegans*, à cause d'une consommation de mémoire trop importante, soulignant ainsi les difficultés de cet outil à passer à l'échelle sur des génomes de taille importante.

La plupart des méthodes de correction hybride, quant à elles, sont parvenues à fournir des corrections satisfaisantes. En effet, sur les douze outils évalués, seuls EC-Tools, Hercules, LSC, et NaS (pour les raisons mentionnées Section 5.4.2) ont produit des *reads* corrigés affichant plus de 1% d'erreurs sur l'ensemble des jeux de données. Les autres méthodes ont toutes permis de fournir un grand nombre de *reads* corrigés de haute qualité. Les taux d'erreurs des *reads* corrigés du jeu de données *C. elegans* se sont cependant révélés relativement plus élevés que ceux des *reads* corrigés des autres jeux de données, à cause de la complexité supérieure de ce génome.

Ces résultats soulignent ainsi que l'auto-correction n'est globalement pas adaptée au traitement de jeux de données disposant de faibles profondeurs de séquençage (20x) et de taux d'erreurs relativement élevés (18,6%). La correction hybride est donc préférable pour le traitement de ce type de données.

7.2.2 Données réelles

7.2.2.1 Qualité des *reads* longs corrigés

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
CoLoRMap	Nombre de *reads*	36 422	25 635	72 017	184 092
	Nombre de bases	141 415 030	114 722 711	165 218 405	418 509 369
	Longueur moyenne (bp)	3 882	4 475	2 294	2 273
	Reads alignés (%)	99,9973	100,0000	99,7403	99,9473
	Identité moyenne (%)	99,5079	99,5751	99,6958	98,6614
	Couverture du génome (%)	100,0000	100,0000	99,1528	96,4079
	Temps	3 h 41 min	2 h 01 min	10 h 44 min	91 h 18 min
	Mémoire (Mo)	13 028	12 121	18 241	31 349
ECTools	Nombre de *reads*	6 882	1 639	21 780	18
	Nombre de bases	36 892 324	6 261 214	116 347 153	54 323
	Longueur moyenne (bp)	5 360	3 820	5 341	3 017
	Reads alignés (%)	100,0000	100,0000	100,0000	100,0000
	Identité moyenne (%)	98,6844	98,4789	98,6497	98,4596
	Couverture du génome (%)	95,3957	26,9638	87,2324	0,016
	Temps	26 min	12 min	3 h 13 min	81 h 25 min
	Mémoire (Mo)	862	922	2 256	5 023
FMLRC	Nombre de *reads*	89 011	22 270	205 923	363 500
	Nombre de bases	390 735 419	134 402 291	1 185 455 434	2 063 059 647
	Longueur moyenne (bp)	4 389	6 035	5 756	5 675
	Reads alignés (%)	27,7741	98,2937	31,7497	78,1986
	Identité moyenne (%)	99,6779	99,9252	96,7164	96,5611
	Couverture du génome (%)	100,0000	100,0000	99,8120	99,9932
	Temps	2 h 01 min	43 min	6 h 15 min	7 h 54 min
	Mémoire (Mo)	449	384	876	7 141

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
HALC[1]	Nombre de *reads*	42 188	24 159	135 050	-
	Nombre de bases	189 757 075	130 674 260	255 641 564	-
	Longueur moyenne (bp)	4 497	5 408	1 892	-
	Reads alignés (%)	99,6444	100,0000	99,5091	-
	Identité moyenne (%)	99,8345	99,9273	99,2933	-
	Couverture du génome (%)	100,0000	100,0000	99,1874	-
	Temps	47 h 41 min	2 h 14 min	2 h 56 min	-
	Mémoire (Mo)	10 577	2 237	2 329	-
Hercules	Nombre de *reads*	89 011	22 270	205 923	363 500
	Nombre de bases	383 188 837	134 641 551	1 173 722 512	2 012 600 582
	Longueur moyenne (bp)	4 304	6 045	5 699	5 536
	Reads alignés (%)	21,5468	95,815	22,7682	72,8468
	Identité moyenne (%)	81,3432	85,7397	71,4998	82,3142
	Couverture du génome (%)	100,0000	100,0000	99,7669	99,9776
	Temps	16 h 53 min	10 h 15 min	12 h 13 min	20 h 36 min
	Mémoire (Mo)	4 438	2 104	19 885	32 000
HG-CoLoR	Nombre de *reads*	25 536	21 986	76 193	305 777
	Nombre de bases	284 883 716	133 956 298	512 438 767	1 567 516 029
	Longueur moyenne (bp)	11 156	6 092	6 725	5 126
	Reads alignés (%)	99,9883	100,0000	99,7126	99,8653
	Identité moyenne (%)	99,9760	99,9589	99,7176	99,5430
	Couverture du génome (%)	100,0000	100,0000	99,5341	99,9655
	Temps	1 h 34 min	59 min	8 h 51 min	83 h 10 min
	Mémoire (Mo)	3 750	1 508	11 575	19 836
Jabba	Nombre de *reads*	17 483	22 086	37 703	270 929
	Nombre de bases	179 366 684	128 040 222	243 374 292	433 372 687
	Longueur moyenne (bp)	10 259	5 797	6 455	1 599
	Reads alignés (%)	99,9714	99,9457	98,5386	97,2901
	Identité moyenne (%)	99,9226	99,9687	99,8889	99,9430
	Couverture du génome (%)	99,8170	99,4309	93,3275	95,2852
	Temps	2 min	2 min	7 min	59 min
	Mémoire (Mo)	1 217	1 220	1 217	13 362
LoRDEC	Nombre de *reads*	50 776	31 728	196 091	167 699
	Nombre de bases	175 140 999	125 542 707	220 706 773	221 676 779
	Longueur moyenne (bp)	3 449	3 956	1 125	1 321
	Reads alignés (%)	99,8779	96,6118	98,5094	96,0137
	Identité moyenne (%)	99,9448	99,9300	98,8168	98,2718
	Couverture du génome (%)	100,0000	100,0000	98,8934	85,0782
	Temps	16 min	13 min	1 h 09 min	1 h 15 min
	Mémoire (Mo)	436	458	797	2 373
LSC[2]	Nombre de *reads*	-	16 744	10 054	-
	Nombre de bases	-	93 660 098	36 116 073	-
	Longueur moyenne (bp)	-	5 593	3 592	-
	Reads alignés (%)	-	100,0000	93,2664	-
	Identité moyenne (%)	-	91,1945	88,5159	-
	Couverture du génome (%)	-	100,0000	88,2405	-
	Temps	-	1 h 10 min	8 h 58 min	-
	Mémoire (Mo)	-	1 832	1 431	-
Nanocorr[3]	Nombre de *reads*	24 105	21 764	66 953	-
	Nombre de bases	173 666 898	128 375 708	231 317 559	-
	Longueur moyenne (bp)	7 204	5 898	3 454	-
	Reads alignés (%)	99,7884	99,9954	98,8873	-
	Identité moyenne (%)	98,3639	99,1417	97,1970	-
	Couverture du génome (%)	100,0000	100,0000	99,5137	-
	Temps	22 h 28 min	5 h 48 min	158 h 53 min	-
	Mémoire (Mo)	616	354	3 128	-

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
NaS[4]	Nombre de *reads*	24 063	21 818	71 793	-
	Nombre de bases	212 707 189	172 918 739	426 326 355	-
	Longueur moyenne (bp)	8 839	7 925	5 938	-
	Reads alignés (%)	100,0000	100,0000	99,8231	-
	Identité moyenne (%)	99,9900	99,9793	99,8985	-
	Couverture du génome (%)	100,0000	100,0000	98,7695	-
	Temps	94 h 19 min	72 h 02 min	$>$ 390 h	-
	Mémoire (Mo)	2 099	2 099	$>$ 16 jours	-
Proovread	Nombre de *reads*	38 432	25 338	91 055	364 244
	Nombre de bases	155 655 071	125 795 081	160 204 364	1 329 812 416
	Longueur moyenne (bp)	4 050	4 964	1 759	3 650
	Reads alignés (%)	100,0000	100,0000	99,7156	99,7194
	Identité moyenne (%)	99,9686	99,9633	99,8979	99,6065
	Couverture du génome (%)	100,0000	100,0000	99,1089	99,9358
	Temps	3 h 25 min	2 h 14 min	13 h 42 min	119 h 48 min
	Mémoire (Mo)	10 618	7 166	8 709	20 254
Canu[5]	Nombre de *reads*	8 632	17 223	-	340 826
	Nombre de bases	80 666 491	121 934 133	-	1 843 278 326
	Longueur moyenne (bp)	9 345	7 079	-	5 408
	Reads alignés (%)	99,9421	99,9942	-	76,7204
	Identité moyenne (%)	94,5919	93,8627	-	90,6784
	Couverture du génome (%)	99,7861	100,0000	-	99,9783
	Temps	31 min	35 min	-	14 h 19 min
	Mémoire (Mo)	3 015	3 264	-	9 178
CONSENT	Nombre de *reads*	16 928	18 143	24 862	256 568
	Nombre de bases	183 088 372	124 479 635	178 658 682	1 475 025 831
	Longueur moyenne (bp)	10 815	6 861	7 186	5 749
	Reads alignés (%)	99,4506	100,0000	96,7702	96,7615
	Identité moyenne (%)	91,9470	94,2336	76,7265	91,6888
	Couverture du génome (%)	100,0000	100,0000	98,1075	99,7682
	Temps	48 min	22 min	40 min	9 h 50 min
	Mémoire (Mo)	5 150	2 239	14 663	20 382
Daccord[6]	Nombre de *reads*	53 926	26 502	-	-
	Nombre de bases	174 962 080	119 130 415	-	-
	Longueur moyenne (bp)	3 244	4 495	-	-
	Reads alignés (%)	97,1906	99,9170	-	-
	Identité moyenne (%)	93,2546	96,7690	-	-
	Couverture du génome (%)	100,0000	100,0000	-	-
	Temps	43 min	20 min	-	-
	Mémoire (Mo)	25 801	6 515	-	-
FLAS[7]	Nombre de *reads*	18 658	16 775	34 483	-
	Nombre de bases	165 473 591	100 467 965	220 777 235	-
	Longueur moyenne (bp)	8 868	5 989	6 402	-
	Reads alignés (%)	99,8553	99,8808	84,1197	-
	Identité moyenne (%)	91,6074	91,4421	77,1713	-
	Couverture du génome (%)	100,0000	100,0000	98,6900	-
	Temps	32 min	8 min	39 min	-
	Mémoire (Mo)	3 015	1 611	7 398	-
LoRMA	Nombre de *reads*	333 041	108 980	48 824	38 121
	Nombre de bases	76 354 713	17 947 836	11 246 944	10 277 321
	Longueur moyenne (bp)	229	164	230	269
	Reads alignés (%)	99,9168	99,9128	57,6397	91,5506
	Identité moyenne (%)	98,0710	98,6822	95,2610	97,6599
	Couverture du génome (%)	66,4788	25,0530	3,6315	1,6269
	Temps	29 min	12 min	1 h 35 min	1 h 13 min
	Mémoire (Mo)	31 575	32 117	1 505	32 275

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
MECAT[8]	Nombre de *reads*	16 811	16 089	14 740	-
	Nombre de bases	154 436 057	106 757 446	83 553 890	-
	Longueur moyenne (bp)	9 186	6 635	5 668	-
	Reads alignés (%)	100,0000	100,0000	99,4573	-
	Identité moyenne (%)	91,4676	92,2218	80,0763	-
	Couverture du génome (%)	100,0000	100,0000	92,6533	-
	Temps	23 min	4 min	14 min	-
	Mémoire (Mo)	2 978	1 681	7 374	-

TABLE 7.2 – **Comparaison de la qualité des *reads* longs corrigés par les différentes méthodes de l'état de l'art, sur les jeux de données réelles de la Table 5.3.** [1] HALC a reporté une erreur lors de la correction du jeu de données *C. elegans*. [2] LSC n'est pas parvenu à corriger le jeu de données *A. baylyi*, et n'a pas été lancé sur le jeu de données *C. elegans*, au vu de ses mauvais résultats et de son important temps d'exécution sur le jeu de données simulées *C. elegans* de la table 7.1. [3] Nanocorr n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [4] NaS n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. De plus, sur le jeu de données *S. cerevisiae*, les *reads* longs corrigés ont été récupérés sur le site du Génoscope, pour les mêmes raisons. [5] Canu a reporté une erreur lors de la correction du jeu de données *S. cerevisiae*. [6] Daccord n'a pas pu corriger les jeux de données *S. cerevisiae* et *C. elegans*, l'étape d'alignement avec DALIGNER consommant une quantité de mémoire trop importante. [7] FLAS a reporté une erreur lors de la correction du jeu de données *C. elegans*. [8] MECAT a reporté une erreur lors de la correction du jeu de données *C. elegans*.

Sur ces jeux de données plus bruitées, mais disposant également d'une couverture plus importante, aucune des méthodes d'auto-correction n'est parvenue à fournir de résultats satisfaisants. En effet, les *reads* corrigés par MECAT ont atteint, au plus, une identité moyenne de 92,2218% sur le jeu de données *E. coli*, affichant à la base un taux d'erreurs de 20,54%. Les *reads* corrigés par Daccord ont, quant à eux, atteint une identité moyenne d'au plus 96,7690%, sur ce même jeu de données. Comme lors des expériences sur données simulées de la Section 7.2.1, Daccord n'a pas permis la correction du jeu de données *C. elegans*, à cause d'une consommation de mémoire trop importante. Cependant, Daccord n'a pas non plus permis la correction du jeu de données *S. cerevisiae*, la profondeur de séquençage de ce dernier étant nettement supérieure à celle du jeu de données simulées, et causant également une consommation de mémoire trop importante. LoRMA a produit les *reads* corrigés affichant la plus haute identité, atteignant jusqu'à 99,6822% sur le jeu de données *E. coli*. Cependant, les *reads* corrigés par LoRMA ont également affiché des longueurs moyennes extrêmement faibles, atteignant au plus 269 paires de bases sur le jeu de données *C. elegans*.

Une fois encore, les méthodes de correction hybride, ECTools, Hercules, et LSC exclues, sont parvenues à fournir des corrections satisfaisantes, réduisant nettement les taux d'erreurs des *reads* pour l'ensemble des jeux de données. La plupart de ces méthodes a ainsi pu être appliquée aussi bien à des génomes bactériens qu'au génome eucaryote de *C. elegans*. L'ensemble des outils n'est cependant pas parvenu à passer à l'échelle sur ce dernier. En effet, HALC a reporté une erreur durant le traitement, tandis que Nanocorr et NaS n'ont pas été utilisés, au vu de leurs temps d'exécution trop importants. De plus, sur le jeu de données *S. cerevisiae*, affichant initialement un taux d'erreurs de plus de 44%, seuls HG-CoLoR et NaS sont parvenus

à produire un grand nombre de bases corrigées, provenant de *reads* longs et de haute qualité.

Ces résultats soulignent ainsi que, malgré d'importantes profondeurs de séquençage (jusqu'à 106x pour le jeu de données *A. baylyi*), l'auto-correction n'est pas adaptée au traitement de *reads* affichant des taux d'erreurs supérieurs à 20%. La correction hybride confirme donc son intérêt pour le traitement de données extrêmement bruitées, telles que les *reads* ONT étudiés ici, séquencés à l'aide des premières versions des chimies de cette technologie.

7.2.2.2 Qualité des assemblages

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
CoLoRMap	Nombre de contigs	1	1	71	1 246
	Nombre de contigs alignés	1	1	68	1 237
	Nombre de *breakpoints*	54	40	47	617
	NGA50 (bp)	3 588 032	4 657 185	204 052	58 411
	NGA75 (bp)	3 588 032	4 657 185	88 468	11 122
	Couverture du génome (%)	96,4454	98,5239	87,1140	76,6546
ECTools[1]	Nombre de contigs	101	-	313	-
	Nombre de contigs alignés	101	-	313	-
	Nombre de *breakpoints*	0	-	0	-
	NGA50 (bp)	12 654	-	7 294	-
	NGA75 (bp)	8 579	-	7 294	-
	Couverture du génome (%)	55,6888	-	48,0243	-
FMLRC	Nombre de contigs	1	1	54	1 047
	Nombre de contigs alignés	1	1	54	1 042
	Nombre de *breakpoints*	21	3	69	713
	NGA50 (bp)	3 673 304	4 642 134	378 469	91 545
	NGA75 (bp)	3 673 304	4 642 134	186 053	40 293
	Couverture du génome (%)	96,8804	100,0000	92,2118	82,9120
HALC[2]	Nombre de contigs	2	1	82	-
	Nombre de contigs alignés	2	1	82	-
	Nombre de *breakpoints*	36	17	75	-
	NGA50 (bp)	2 420 463	4 644 236	313 785	-
	NGA75 (bp)	1 207 743	4 644 236	146 613	-
	Couverture du génome (%)	97,3183	99,3693	92,0039	-
Hercules	Nombre de contigs	1	1	174	1 356
	Nombre de contigs alignés	1	1	171	1 348
	Nombre de *breakpoints*	49	63	10	425
	NGA50 (bp)	3 537 383	4 655 108	3 384	36 090
	NGA75 (bp)	3 537 383	4 655 108	3 384	690
	Couverture du génome (%)	97,4830	97,9754	29,1838	66,7194
HG-CoLoR	Nombre de contigs	2	1	53	989
	Nombre de contigs alignés	2	1	53	988
	Nombre de *breakpoints*	5	5	72	461
	NGA50 (bp)	3 595 353	4 643 849	470 355	96 058
	NGA75 (bp)	3 595 353	4 643 849	212 761	41 829
	Couverture du génome (%)	99,9182	99,9739	95,3867	82,6239

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Jabba	Nombre de contigs	17	47	176	880
	Nombre de contigs alignés	17	47	176	880
	Nombre de *breakpoints*	1	2	2	15
	NGA50 (bp)	216 570	133 278	45 205	2 046
	NGA75 (bp)	184 851	61 014	6 507	2 046
	Couverture du génome (%)	93,7381	95,0742	72,0151	30,0434
LoRDEC	Nombre de contigs	2	1	66	21
	Nombre de contigs alignés	2	1	66	16
	Nombre de *breakpoints*	73	71	126	9
	NGA50 (bp)	3 010 005	4 659 557	361 726	589
	NGA75 (bp)	3 010 005	4 659 557	178 604	589
	Couverture du génome (%)	94,6755	97,7848	90,0706	0,0854
LSC[3]	Nombre de contigs	-	7	2	-
	Nombre de contigs alignés	-	7	2	-
	Nombre de *breakpoints*	-	42	0	-
	NGA50 (bp)	-	1 214 279	5 393	-
	NGA75 (bp)	-	548 948	5 393	-
	Couverture du génome (%)	-	98,6104	0,1190	-
Nanocorr[4]	Nombre de contigs	1	1	170	-
	Nombre de contigs alignés	1	1	170	-
	Nombre de *breakpoints*	41	33	26	-
	NGA50 (bp)	3 647 559	4 662 932	84 048	-
	NGA75 (bp)	3 647 559	4 662 932	44 028	-
	Couverture du génome (%)	98,2877	99,1316	89,5144	-
NaS[5]	Nombre de contigs	1	1	159	-
	Nombre de contigs alignés	1	1	159	-
	Nombre de *breakpoints*	2	6	17	-
	NGA50 (bp)	3 600 775	4 641 680	85 205	-
	NGA75 (bp)	3 600 775	4 641 680	43 313	-
	Couverture du génome (%)	99,9986	99,9674	89,7248	-
Proovread	Nombre de contigs	3	1	77	1 095
	Nombre de contigs alignés	3	1	74	1 086
	Nombre de *breakpoints*	60	48	48	509
	NGA50 (bp)	1 304 077	4 658 520	176 285	80 171
	NGA75 (bp)	1 304 077	4 658 520	80 275	31 114
	Couverture du génome (%)	92,7046	98,4384	86,0619	79,6994
Canu[6]	Nombre de contigs	19	1	-	1 272
	Nombre de contigs alignés	19	1	-	1 259
	Nombre de *breakpoints*	3	4	-	512
	NGA50 (bp)	2 146 043	4 457 202	-	54 194
	NGA75 (bp)	99 247	4 457 202	-	416
	Couverture du génome (%)	93,6289	99,9802	-	76,9836
CONSENT	Nombre de contigs	1	1	186	1 279
	Nombre de contigs alignés	1	1	185	1 268
	Nombre de *breakpoints*	23	14	18	624
	NGA50 (bp)	3 494 031	4 519 816	3 326	49 620
	NGA75 (bp)	3 494 031	4 519 816	3 326	1 606
	Couverture du génome (%)	99,1676	99,7372	36,9326	73,1300

Outil	Métrique	Jeu de données			
		A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
Daccord[7]	Nombre de contigs	2	6	-	-
	Nombre de contigs alignés	2	6	-	-
	Nombre de *breakpoints*	4	1	-	-
	NGA50 (bp)	3 472 403	1 140 945	-	-
	NGA75 (bp)	3 472 403	995 717	-	-
	Couverture du génome (%)	100,0000	99,9572	-	-
FLAS[8]	Nombre de contigs	2	19	114	-
	Nombre de contigs alignés	2	19	114	-
	Nombre de *breakpoints*	32	36	8	-
	NGA50 (bp)	1 610 377	307 489	2 974	-
	NGA75 (bp)	1 610 377	210 170	2 974	-
	Couverture du génome (%)	96,9851	99,3639	21,8854	-
LoRMA[9]	Nombre de contigs	1	-	-	-
	Nombre de contigs alignés	1	-	-	-
	Nombre de *breakpoints*	0	-	-	-
	NGA50 (bp)	4 949	-	-	-
	NGA75 (bp)	4 949	-	-	-
	Couverture du génome (%)	0,1406	-	-	-
MECAT[10]	Nombre de contigs	1	7	102	-
	Nombre de contigs alignés	1	7	102	-
	Nombre de *breakpoints*	3	1	2	-
	NGA50 (bp)	3 335 596	843 326	3 163	-
	NGA75 (bp)	3 335 596	610 826	3 163	-
	Couverture du génome (%)	99,9985	99,9204	15,1658	-

TABLE 7.3 – **Comparaison de la qualité des assemblages générés à partir des *reads* longs corrigés par les différentes méthodes de l'état de l'art, sur les jeux de données réelles de la Table 5.3.** [1] Les *reads* longs corrigés par ECTools n'ont pas pu être assemblés pour les jeux de données *E. coli* et *C. elegans*.[2] HALC a reporté une erreur lors de la correction du jeu de données *C. elegans*. [3] LSC n'est pas parvenu à corriger le jeu de données *A. baylyi*, et n'a pas été lancé sur le jeu de données *C. elegans*, au vu de ses mauvais résultats et de son important temps d'exécution sur le jeu de données simulées *C. elegans* de la table 7.1. [4] Nanocorr n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [5] NaS n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [6] Canu a reporté une erreur lors de la correction du jeu de données *S. cerevisiae*. [7] Daccord n'a pas pu corriger les jeux de données *S. cerevisiae* et *C. elegans*, l'étape d'alignement avec DALIGNER consommant une quantité de mémoire trop importante. [8] FLAS a reporté une erreur lors de la correction du jeu de données *C. elegans*. [9] Les *reads* longs corrigés par LoRMA n'ont pas pu être assemblés pour les jeux de données *E. coli*, *S. cerevisiae* et *C. elegans*. [10] MECAT a reporté une erreur lors de la correction du jeu de données *C. elegans*.

Malgré leurs taux d'erreurs élevés, les *reads* longs corrigés par CONSENT et par MECAT ont pu être assemblés en un unique contig pour le jeu de données *A. baylyi*. Sur ce même jeu de données, les *reads* longs corrigés par Daccord ont également permis de générer un assemblage satisfaisant, contenant toute l'information du génome au sein d'un seul contig. Sur le jeu de données *E. coli*, les *reads* longs corrigés par CONSENT et par Canu ont également permis de générer des assemblages composés d'un unique contig. Cependant, sur les autres jeux de données, plus bruitées,

et provenant de génomes plus complexes, aucune méthode d'auto-correction n'est parvenue à produire des *reads* longs corrigés pouvant s'assembler de façon satisfaisante.

Sur les jeux de données *A. baylyi* et *E. coli*, la grande majorité des méthodes de correction hybride a cependant permis de produire des *reads* longs corrigés pouvant générer des assemblages satisfaisants. En effet, seuls ECTools, Jabba et LSC ont produit des assemblages de mauvaise qualité, et composés de nombreux contigs. Comme pour les méthodes d'auto-correction, les *reads* corrigés par Hercules, malgré leurs importants taux d'erreurs, ont tout de même permis de générer des assemblages composés d'un contig unique.

Sur les jeux de données des génomes plus complexes de *S. cerevisiae* et de *C. elegans*, bien que plus faible que pour les génomes bactériens, la qualité des assemblages s'est tout de même révélée relativement satisfaisante. Hormis Hercules et les méthodes précédemment mentionnées, l'ensemble des outils de correction hybride a en effet produit des *reads* longs corrigés permettant de générer des assemblages couvrant d'importantes proportions des génomes de référence. Cette couverture s'est cependant révélée plus faible sur le jeu de données *C. elegans*, au vu de la faible profondeur de séquençage (20x) des *reads* longs originaux. De plus, sur ce jeu de données, les *reads* corrigés par LoRDEC ont généré un assemblage de très mauvaise qualité, couvrant moins de 1% du génome, comme explicité Section 5.4.3.2.

Ces résultats soulignent ainsi que, bien qu'elles produisent des *reads* longs corrigés affichant d'importants taux d'erreurs, les méthodes d'auto-correction peuvent tout de même être utilisées dans l'optique de générer des assemblages bactériens de haute qualité, à partir de *reads* longs très bruités. Pour des jeux de données provenant de génomes plus complexes, les méthodes de correction hybride permettent cependant de produire des *reads* longs corrigés pouvant générer des assemblages de meilleure qualité, même à partir de *reads* extrêmement bruités, affichant des taux d'erreurs de plus de 44%.

7.3 Jeux de données du Chapitre 6

Pour les expériences présentées ici, nous excluons les méthodes de correction hybride suivantes :

- ECTools, au vu de ses mauvais résultats lors des expériences précédentes.
- Hercules, au vu de ses mauvais résultats lors des expériences précédentes, et de ses importants temps d'exécution.
- LSC, au vu de ses mauvais résultats lors des expériences précédentes.
- Nanocorr, au vu de ses temps d'exécution trop importants.
- NaS, au vu de ses temps d'exécution trop importants.

De plus, Proovread est également exclus de la comparaison sur données réelles, puisque n'ayant pas pu être installé sur la machine ayant servi à réaliser les expériences.

7.3.1 Données simulées

7.3.1.1 Profondeur de séquençage de 30x

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
CoLoRMap	Nombre de bases	134 398 575	343 101 257	1 197 628 902
	Taux d'erreurs (%)	0,1137	0,3183	0,8955
	Délétions	14 929	346 678	1 900 447
	Insertions	36 705	291 959	1 036 546
	Substitutions	92 329	449 592	7 663 104
	Rappel (%)	99,9881	99,9135	99,9165
	Précision (%)	99,8880	99,6860	99,1230
	Reads trimmés ou *splittés*	31	845	8 587
	Longueur manquante moyenne (bp)	3 729,9	4 242,4	5 180,3
	Reads étendus	16 644	42 853	168 676
	Longueur moyenne d'extension (bp)	305,5	305,5	252,3
	Reads de mauvaise qualité	13	65	208
	Reads trop courts	269	1 606	194 369
	Ratio longueur homopolymers	1,0000	1,0071	0,9967
	Temps	1 h 33 min	4 h 36 min	150 h 21 min
	Mémoire (Mo)	13 097	14 243	32 267
FMLRC	Nombre de bases	131 066 466	348 205 441	2 820 835 418
	Taux d'erreurs (%)	0,0320	0,2447	1,4161
	Délétions	11 502	273 645	15 775 123
	Insertions	16 155	427 404	16 783 150
	Substitutions	8 752	196 210	10 932 609
	Rappel (%)	99,9919	99,9032	99,6795
	Précision (%)	99,9686	99,7579	98,6018
	Reads trimmés ou *splittés*	2	44	2 090
	Longueur manquante moyenne (bp)	17	66	28,4
	Reads étendus	0	3	106
	Longueur moyenne d'extension (bp)	0	0	31,3
	Reads de mauvaise qualité	41	151	1 121
	Reads trop courts	0	0	0
	Ratio longueur homopolymers	1,0000	1,0000	0,9915
	Temps	45 min	1 h 59 min	11 h 55 min
	Mémoire (Mo)	400	892	7 937
HALC	Nombre de bases	130 839 109	347 722 386	2 819 386 712
	Taux d'erreurs (%)	0,1565	0,3611	1,0897
	Délétions	121 296	444 792	9 586 042
	Insertions	40 749	403 321	12 999 176
	Substitutions	23 408	405 608	9 627 633
	Rappel (%)	99,9799	99,9387	99,7543
	Précision (%)	99,8466	99,6455	98,9252
	Reads trimmés ou *splittés*	1 013	2 812	31 158
	Longueur manquante moyenne (bp)	140,8	151,8	101,8
	Reads étendus	6	47	599
	Longueur moyenne d'extension (bp)	28,8	28,8	40,2
	Reads de mauvaise qualité	41	136	1 195
	Reads trop courts	0	1	2
	Ratio longueur homopolymers	1,0000	1,0000	1,0000
	Temps	1 h 17 min	1 h 53 min	9 h 30 min
	Mémoire (Mo)	1 714	1 892	2 853

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
HG-CoLoR	Nombre de bases	131 346 207	346 708 590	2 794 826 522
	Taux d'erreurs (%)	0,0726	0,5115	1,1664
	Délétions	39 908	1 342 278	26 965 036
	Insertions	49 798	550 820	9 324 074
	Substitutions	5 741	162 266	1 509 055
	Rappel (%)	99,9986	99,9592	99,9104
	Précision (%)	99,9279	99,4937	98,8449
	Reads trimmés ou *splittés*	626	4 141	59 301
	Longueur manquante moyenne (bp)	178,9	331,7	214,8
	Reads étendus	8 293	16 823	130 681
	Longueur moyenne d'extension (bp)	30,3	30,3	29,4
	Reads de mauvaise qualité	0	60	435
	Reads trop courts	0	7	2
	Ratio longueur homopolymers	1,0000	1,0012	1,0000
	Temps	1 h 20 min	7 h 20 min	108 h 26 min
	Mémoire (Mo)	1 538	3 656	27 212
Jabba	Nombre de bases	133 959 455	340 101 149	2 463 694 280
	Taux d'erreurs (%)	0,0771	0,1067	0,2319
	Délétions	80 450	209 683	4 332 786
	Insertions	15 541	155 778	1 242 983
	Substitutions	573	33 825	236 640
	Rappel (%)	99,9981	99,9975	99,9913
	Précision (%)	99,9237	99,8941	99,7702
	Reads trimmés ou *splittés*	2 037	10 781	148 923
	Longueur manquante moyenne (bp)	2 601,5	1 992,5	2 805,6
	Reads étendus	15 416	37 196	247 049
	Longueur moyenne d'extension (bp)	638,6	638,6	947,3
	Reads de mauvaise qualité	2	77	367
	Reads trop courts	874	5 946	69 905
	Ratio longueur homopolymers	1,0000	1,0000	1,0000
	Temps	2 min	5 min	43 min
	Mémoire (Mo)	1 220	1 215	13 362
LoRDEC	Nombre de bases	131 052 792	348 420 743	2 823 537 319
	Taux d'erreurs (%)	0,0695	0,3990	1,2710
	Délétions	32 861	297 482	8 411 355
	Insertions	35 620	694 323	19 478 144
	Substitutions	24 234	480 342	10 963 740
	Rappel (%)	99,9831	99,8123	99,4191
	Précision (%)	99,9328	99,6093	98,7441
	Reads trimmés ou *splittés*	181	2 354	56 726
	Longueur manquante moyenne (bp)	63,7	79,7	126,2
	Reads étendus	2	2	149
	Longueur moyenne d'extension (bp)	25,5	25,5	38,9
	Reads de mauvaise qualité	22	97	558
	Reads trop courts	0	0	3
	Ratio longueur homopolymers	1,0000	1,0000	0,9979
	Temps	12 min	35 min	11 h 30 min
	Mémoire (Mo)	460	799	2 320

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
Proovread	Nombre de bases	130 360 408	342 457 860	2 703 725 143
	Taux d'erreurs (%)	0,1615	0,2365	0,4325
	Délétions	169 197	845 704	20 509 633
	Insertions	1 309	30 583	189 378
	Substitutions	6 636	43 845	654 789
	Rappel (%)	99,9953	99,9912	99,9711
	Précision (%)	99,8409	99,7668	99,5738
	Reads trimmés ou *splittés*	8 571	29 179	153 272
	Longueur manquante moyenne (bp)	44,1	85,6	227,3
	Reads étendus	0	3	45
	Longueur moyenne d'extension (bp)	0	0	87,9
	Reads de mauvaise qualité	0	135	1 899
	Reads trop courts	29	1 202	15 249
	Ratio longueur homopolymers	1,0000	1,0000	1,0000
	Temps	1 h 59 min	5 h 37 min	85 h 23 min
	Mémoire (Mo)	8 368	16 777	29 934
Canu	Nombre de bases	129 533 580	226 459 133	2 773 456 245
	Taux d'erreurs (%)	0,4156	1,1052	0,5008
	Délétions	247 325	803 320	6 013 667
	Insertions	237 224	1 371 782	6 130 158
	Substitutions	19 506	145 574	517 864
	Rappel (%)	99,7647	99,1766	99,7103
	Précision (%)	99,5887	98,9036	99,5040
	Reads trimmés ou *splittés*	9 422	33 164	170 416
	Longueur manquante moyenne (bp)	128,9	1 762,6	126,5
	Reads étendus	55	51	1 100
	Longueur moyenne d'extension (bp)	26,7	28,5	28,1
	Reads de mauvaise qualité	0	41	393
	Reads trop courts	3	509	147
	Ratio longueur homopolymers	1,0116	1,0040	0,9990
	Temps	19 min	29 min	9 h 09 min
	Mémoire (Mo)	4 613	3 681	6 921
CONSENT	Nombre de bases	129 647 002	344 457 307	2 789 537 501
	Taux d'erreurs (%)	0,3142	0,4091	0,7902
	Délétions	254 833	853 156	12 522 232
	Insertions	134 538	456 318	8 794 630
	Substitutions	18 772	112 883	2 296 195
	Rappel (%)	99,9430	99,9321	99,8773
	Précision (%)	99,6906	99,5971	99,2204
	Reads trimmés ou *splittés*	14 681	38 872	309 728
	Longueur manquante moyenne (bp)	86,7	83,8	80,8
	Reads étendus	0	5	107
	Longueur moyenne d'extension (bp)	0	0	83,1
	Reads de mauvaise qualité	0	41	424
	Reads trop courts	0	0	0
	Ratio longueur homopolymers	1,0000	0,9967	1,0077
	Temps	17 min	46 min	9 h 36 min
	Mémoire (Mo)	2 390	5 523	21 819

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
Daccord[1]	Nombre de bases	131 020 333	347 992 121	-
	Taux d'erreurs (%)	0,0248	0,1259	-
	Délétions	4 326	73 057	-
	Insertions	13 333	123 760	-
	Substitutions	15 965	265 120	-
	Rappel (%)	99,9965	99,9874	-
	Précision (%)	99,9757	99,8762	-
	Reads trimmés ou *splittés*	5	102	-
	Longueur manquante moyenne (bp)	478,8	188,5	-
	Reads étendus	0	5	-
	Longueur moyenne d'extension (bp)	0	0	-
	Reads de mauvaise qualité	0	47	-
	Reads trop courts	0	21	-
	Ratio longueur homopolymers	1,0000	1,0000	-
	Temps	14 min	1 h 19 min	-
	Mémoire (Mo)	6 813	31 798	-
FLAS	Nombre de bases	129 692 582	344 252 752	2 728 691 582
	Taux d'erreurs (%)	0,2720	0,3272	0,7613
	Délétions	308 007	967 396	22 672 322
	Insertions	92 042	273 904	7 047 712
	Substitutions	6 952	35 833	1 451 039
	Rappel (%)	99,9291	99,9131	99,8613
	Précision (%)	99,7385	99,6843	99,2541
	Reads trimmés ou *splittés*	1 900	5 082	72 527
	Longueur manquante moyenne (bp)	434	438,3	581,6
	Reads étendus	0	1	578
	Longueur moyenne d'extension (bp)	0	0	161,2
	Reads de mauvaise qualité	5	58	7 480
	Reads trop courts	0	0	24
	Ratio longueur homopolymers	1,0076	0,9972	1,0036
	Temps	12 min	29 min	3 h 07 min
	Mémoire (Mo)	1 639	2 935	10 565
LoRMA	Nombre de bases	2 378 146	14 071 752	32 644 556
	Taux d'erreurs (%)	1,1962	2,1640	3,6960
	Délétions	27 701	474 954	2 058 432
	Insertions	1 631	40 421	123 001
	Substitutions	6 271	35 339	182 811
	Rappel (%)	99,9209	99,8351	99,7269
	Précision (%)	98,8208	97,8564	96,3449
	Reads trimmés ou *splittés*	11 737	35 483	227 760
	Longueur manquante moyenne (bp)	225,6	179,1	158,1
	Reads étendus	0	3	16
	Longueur moyenne d'extension (bp)	0	0	492,7
	Reads de mauvaise qualité	7	485	2 758
	Reads trop courts	63 731	199 074	1 163 973
	Ratio longueur homopolymers	1,0000	1,0000	1,0000
	Temps	10 min	46 min	8 h 19 min
	Mémoire (Mo)	32 155	31 899	31 827

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
MECAT	Nombre de bases	106 842 493	284 618 017	2 083 992 906
	Taux d'erreurs (%)	0,2569	0,3040	0,3908
	Délétions	238 111	738 020	7 017 472
	Insertions	46 384	154 073	1 559 772
	Substitutions	2 525	16 281	93 949
	Rappel (%)	99,9302	99,9160	99,8903
	Précision (%)	99,7533	99,7072	99,6212
	Reads trimmés ou *splittés*	6 625	16 957	148 957
	Longueur manquante moyenne (bp)	1 261	1 229,9	1 468,4
	Reads étendus	0	0	9
	Longueur moyenne d'extension (bp)	0	0	598,1
	Reads de mauvaise qualité	0	44	188
	Reads trop courts	0	0	0
	Ratio longueur homopolymers	1,0104	1,0043	0,9966
	Temps	2 min	5 min	48 min
	Mémoire (Mo)	1 600	2 907	10 535

TABLE 7.4 – **Comparaison de la qualité de la correction fournie par les différentes méthodes de l'état de l'art, sur les jeux de données simulées de la Table 6.3 disposant d'une profondeur de séquençage de 30x.** [1] Daccord a reporté une erreur dès le lancement de l'étape d'alignement avec DALIGNER, sur le jeu de données *C. elegans*.

Hormis LoRMA et Canu sur le jeu de données *E. coli*, l'ensemble des méthodes d'auto-correction ont ici permis de réduire les taux d'erreurs de tous les jeux de données en dessous de 1%, malgré une profondeur de séquençage relativement faible. De la même façon, toutes les méthodes de correction hybride ont également permis de produire des *reads* longs corrigés affichant des taux d'erreurs inférieurs à 1% sur les jeux de données *E. coli* et *S. cerevisiae*. Cependant, contrairement à ce qui a été observé dans les expériences de la Section 7.2.1, les méthodes d'auto-correction ont, ici, fourni de meilleurs résultats que la plupart des méthodes de correction hybride sur le jeu de données *C. elegans*. En effet, sur ce jeu de données, seuls CoLoRMap, Jabba et Proovread sont parvenus à réduire le taux d'erreurs des *reads* à moins de 1%. Ces résultats peuvent notamment être expliqués par le fait que les répétitions, plus nombreuses dans le génome eucaryote de *C. elegans*, peuvent être résolues plus aisément par auto-correction, en tirant pleinement profit de l'information portée par la longueur des *reads*.

Les temps d'exécution des différentes méthodes soulignent également un net avantage pour les méthodes d'auto-correction. En effet, hormis Jabba, et LoRDEC sur les jeux de données *E. coli* et *S. cerevisiae*, les méthodes d'auto-correction ont affiché des temps d'exécution globalement inférieurs aux méthodes de correction hybride. En particulier, sur le jeu de données *C. elegans*, la correction hybride a pu demander jusqu'à 150 heures, en utilisant CoLoRMap, tandis que Canu et CONSENT, les méthodes d'auto-correction les plus lentes, n'ont nécessité respectivement que 9 h 09 minutes et 9 h 36 minutes.

Ces résultats soulignent ainsi que l'auto-correction est plus adaptée que la correction hybride, pour le traitement de jeux de données disposant de taux d'erreurs relativement faibles (12%). De plus, les jeux de données étudiés ici ne disposent que d'une profondeur de séquençage de 30x. Ainsi, ces résultats soulignent également que d'importantes profondeurs de séquençage ne sont pas nécessaires aux méthodes d'auto-correction pour produire des résultats de haute qualité.

7.3.1.2 Profondeur de séquençage de 60x

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
CoLoRMap[1]	Nombre de bases	266 210 819	663 617 071	-
	Taux d'erreurs (%)	0,1621	0,6143	-
	Délétions	98 293	1 930 344	-
	Insertions	112 511	1 216 405	-
	Substitutions	199 934	1 036 618	-
	Rappel (%)	99,9631	99,7755	-
	Précision (%)	99,8400	99,3917	-
	Reads trimmés ou *splittés*	185	3 913	-
	Longueur manquante moyenne (bp)	3 787,2	3 746,2	-
	Reads étendus	33 047	82 438	-
	Longueur moyenne d'extension (bp)	297,7	297,7	-
	Reads de mauvaise qualité	33	145	-
	Reads trop courts	653	4 679	-
	Ratio longueur homopolymers	1,0089	0,9946	-
	Temps	3 h 01 min	8 h 08 min	-
	Mémoire (Mo)	19 898	24 375	-
FMLRC	Nombre de bases	261 387 632	695 239 831	5 652 356 620
	Taux d'erreurs (%)	0,0292	0,2469	1,4213
	Délétions	24 753	551 321	31 671 968
	Insertions	31 069	872 824	33 701 759
	Substitutions	17 368	391 349	21 909 721
	Rappel (%)	99,9944	99,9006	99,6764
	Précision (%)	99,9714	99,7557	98,5966
	Reads trimmés ou *splittés*	14	106	4 147
	Longueur manquante moyenne (bp)	17,7	48,5	28,9
	Reads étendus	0	8	242
	Longueur moyenne d'extension (bp)	0	0	28,1
	Reads de mauvaise qualité	85	360	2 315
	Reads trop courts	0	0	0
	Ratio longueur homopolymers	1,0000	1,0000	0,9856
	Temps	1 h 28 min	3 h 57 min	23 h 25 min
	Mémoire (Mo)	403	893	7 937
HALC	Nombre de bases	260 933 848	694 333 491	5 649 091 260
	Taux d'erreurs (%)	0,1522	0,3648	1,0880
	Délétions	233 633	883 877	19 179 388
	Insertions	84 745	822 236	25 917 654
	Substitutions	48 696	827 403	19 226 301
	Rappel (%)	99,9785	99,9354	99,7550
	Précision (%)	99,8509	99,6420	98,9269
	Reads trimmés ou *splittés*	2 033	5 707	61 894
	Longueur manquante moyenne (bp)	147,3	131,6	102
	Reads étendus	11	60	1 284
	Longueur moyenne d'extension (bp)	22,4	22,4	42,6
	Reads de mauvaise qualité	79	329	2 512
	Reads trop courts	0	1	0
	Ratio longueur homopolymers	1,0000	1,0000	0,9852
	Temps	4 h 57 min	4 h 25 min	19 h 10 min
	Mémoire (Mo)	2 747	2 487	5 716

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
HG-CoLoR^2	Nombre de bases	261 879 085	690 178 663	-
	Taux d'erreurs (%)	0,0771	0,5995	-
	Délétions	87 607	2 824 556	-
	Insertions	113 291	1 924 555	-
	Substitutions	14 055	374 714	-
	Rappel (%)	99,9987	99,9433	-
	Précision (%)	99,9234	99,4059	-
	Reads trimmés ou *splittés*	1 569	9 700	-
	Longueur manquante moyenne (bp)	264,1	496,2	-
	Reads étendus	19 498	40 638	-
	Longueur moyenne d'extension (bp)	33,2	33,2	-
	Reads de mauvaise qualité	1	235	-
	Reads trop courts	0	45	-
	Ratio longueur homopolymers	1,0000	1,0000	-
	Temps	2 h 03 min	12 h 23 min	-
	Mémoire (Mo)	2 744	7 297	-
Jabba	Nombre de bases	267 665 882	678 936 284	4 934 623 442
	Taux d'erreurs (%)	0,0716	0,1040	0,2312
	Délétions	146 631	387 538	8 496 067
	Insertions	30 775	340 301	2 534 990
	Substitutions	1 171	63 194	419 150
	Rappel (%)	99,9984	99,9975	99,9916
	Précision (%)	99,9292	99,8967	99,7710
	Reads trimmés ou *splittés*	4 095	21 726	298 954
	Longueur manquante moyenne (bp)	2 507,2	1 996	2 804,2
	Reads étendus	30 786	74 343	493 302
	Longueur moyenne d'extension (bp)	633,2	633,2	949,5
	Reads de mauvaise qualité	1	183	731
	Reads trop courts	1 649	11 922	140 889
	Ratio longueur homopolymers	1,0000	1,0000	1,0000
	Temps	2 min	5 min	49 min
	Mémoire (Mo)	1 220	1 215	13 360
LoRDEC	Nombre de bases	261 360 235	695 578 111	5 657 161 182
	Taux d'erreurs (%)	0,0684	0,3984	1,2731
	Délétions	65 549	613 150	16 882 129
	Insertions	70 097	1 389 483	38 979 226
	Substitutions	47 336	974 285	21 920 160
	Rappel (%)	99,9832	99,8136	99,4201
	Précision (%)	99,9339	99,6100	98,7420
	Reads trimmés ou *splittés*	393	4 657	113 843
	Longueur manquante moyenne (bp)	65,3	78,9	126,3
	Reads étendus	0	2	273
	Longueur moyenne d'extension (bp)	0	0	48
	Reads de mauvaise qualité	35	214	1 263
	Reads trop courts	0	1	11
	Ratio longueur homopolymers	1,0000	1,0000	1,0160
	Temps	20 min	1 h 09 min	23 h 30 min
	Mémoire (Mo)	457	794	2 332

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
Proovread[3]	Nombre de bases	259 647 167	671 307 477	-
	Taux d'erreurs (%)	0,1689	0,2568	-
	Délétions	358 693	2 080 733	-
	Insertions	2 837	56 965	-
	Substitutions	15 499	102 130	-
	Rappel (%)	99,9948	99,9902	-
	Précision (%)	99,8335	99,7467	-
	Reads trimmés ou *splittés*	18 693	68 598	-
	Longueur manquante moyenne (bp)	54,3	148,2	-
	Reads étendus	4	10	-
	Longueur moyenne d'extension (bp)	20,2	20,2	-
	Reads de mauvaise qualité	0	501	-
	Reads trop courts	172	7 658	-
	Ratio longueur homopolymers	1,0000	1,0000	-
	Temps	4 h 07 min	11 h 51 min	-
	Mémoire (Mo)	15 245	23 591	-
Canu	Nombre de bases	218 652 792	599 325 043	5 112 430 638
	Taux d'erreurs (%)	0,7404	0,7919	0,7934
	Délétions	563 365	1 678 564	14 729 573
	Insertions	827 596	2 412 967	20 996 517
	Substitutions	72 521	240 442	1 897 014
	Rappel (%)	99,4781	99,4488	99,4573
	Précision (%)	99,2658	99,2148	99,2131
	Reads trimmés ou *splittés*	27 690	73 001	569 082
	Longueur manquante moyenne (bp)	985,8	858,1	651,3
	Reads étendus	47	176	1 658
	Longueur moyenne d'extension (bp)	29,0	53,7	37,2
	Reads de mauvaise qualité	0	104	941
	Reads trop courts	187	354	1 845
	Ratio longueur homopolymers	1,0158	1,0108	1,0065
	Temps	24 min	1 h 11 min	9 h 30 min
	Mémoire (Mo)	3 674	3 710	7 050
CONSENT	Nombre de bases	259 078 481	689 502 352	5 603 760 432
	Taux d'erreurs (%)	0,1722	0,2918	0,5730
	Délétions	341 549	1 280 545	18 391 216
	Insertions	55 787	582 869	12 737 445
	Substitutions	8 354	115 614	2 826 988
	Rappel (%)	99,9843	99,9481	99,9119
	Précision (%)	99,8306	99,7125	99,4345
	Reads trimmés ou *splittés*	27 380	71 027	571 107
	Longueur manquante moyenne (bp)	71,8	68,1	67,1
	Reads étendus	0	2	187
	Longueur moyenne d'extension (bp)	0	0	122,2
	Reads de mauvaise qualité	0	107	849
	Reads trop courts	0	0	0
	Ratio longueur homopolymers	1,0000	1,0000	0,9653
	Temps	36 min	1 h 46 min	27 h 04 min
	Mémoire (Mo)	4 849	11 325	32 284

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
Daccord[4]	Nombre de bases	261 274 775	694 845 174	-
	Taux d'erreurs (%)	0,0214	0,0400	-
	Délétions	6 353	51 217	-
	Insertions	19 335	132 765	-
	Substitutions	28 842	75 540	-
	Rappel (%)	99,9971	99,9928	-
	Précision (%)	99,9790	99,9606	-
	Reads trimmés ou *splittés*	13	73	-
	Longueur manquante moyenne (bp)	58,8	276,8	-
	Reads étendus	0	4	-
	Longueur moyenne d'extension (bp)	0	0	-
	Reads de mauvaise qualité	0	113	-
	Reads trop courts	0	0	-
	Ratio longueur homopolymers	1,0000	1,0000	-
	Temps	54 min	2 h 26 min	-
	Mémoire (Mo)	18 450	32 190	-
FLAS	Nombre de bases	260 324 873	689 491 882	5 583 925 852
	Taux d'erreurs (%)	0,1547	0,2034	0,3997
	Délétions	378 728	1 273 157	25 708 666
	Insertions	53 059	220 507	5 628 454
	Substitutions	3 141	30 665	1 249 658
	Rappel (%)	99,9546	99,9418	99,9175
	Précision (%)	99,8526	99,8049	99,6104
	Reads trimmés ou *splittés*	403	1 274	30 950
	Longueur manquante moyenne (bp)	182,4	223,4	305,4
	Reads étendus	1	5	588
	Longueur moyenne d'extension (bp)	102	102	156,7
	Reads de mauvaise qualité	4	241	7 506
	Reads trop courts	0	0	14
	Ratio longueur homopolymers	0,9967	1,0000	0,9890
	Temps	38 min	1 h 30 min	10 h 45 min
	Mémoire (Mo)	2 428	4 984	13 682
LoRMA	Nombre de bases	170 581 660	442 516 834	780 732 399
	Taux d'erreurs (%)	0,1285	0,2225	0,6446
	Délétions	163 750	1 197 979	5 515 626
	Insertions	25 117	129 027	369 539
	Substitutions	21 604	106 535	581 292
	Rappel (%)	99,9865	99,9785	99,9547
	Précision (%)	99,8743	99,7812	99,3649
	Reads trimmés ou *splittés*	40 347	108 516	1 154 643
	Longueur manquante moyenne (bp)	1 007	970,3	412,3
	Reads étendus	0	4	13
	Longueur moyenne d'extension (bp)	0	0	685,7
	Reads de mauvaise qualité	4	797	3 575
	Reads trop courts	187 376	522 390	9 256 368
	Ratio longueur homopolymers	1,0000	1,0000	1,0000
	Temps	1 h 39 min	5 h 25 min	31 h 04 min
	Mémoire (Mo)	31 682	31 828	32 104

Outil	Métrique	Jeu de données		
		E. coli	*S. cerevisiae*	*C. elegans*
MECAT	Nombre de bases	232 519 767	616 247 287	4 937 676 257
	Taux d'erreurs (%)	0,1714	0,2088	0,2675
	Délétions	392 085	1 228 519	12 514 560
	Insertions	43 487	168 401	1 985 795
	Substitutions	2 006	14 471	107 294
	Rappel (%)	99,9547	99,9428	99,9258
	Précision (%)	99,8362	99,7996	99,7415
	Reads trimmés ou *splittés*	3 622	9 606	112 978
	Longueur manquante moyenne (bp)	365	380,7	566,8
	Reads étendus	0	0	30
	Longueur moyenne d'extension (bp)	0	0	165
	Reads de mauvaise qualité	0	92	443
	Reads trop courts	0	0	0
	Ratio longueur homopolymers	0,9967	1,0000	1,0000
	Temps	5 min	16 min	2 h 43 min
	Mémoire (Mo)	2 387	4 954	10 563

TABLE 7.5 – **Comparaison de la qualité de la correction fournie par les différentes méthodes de l'état de l'art, sur les jeux de données simulées de la Table 6.3 disposant d'une profondeur de séquençage de 60x.** [1] CoLoRMap n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [2] HG-CoLoR n'a pas été lancé sur le jeu de données *C. elegans* au vu de ces temps d'exécution trop importants. [3] Proovread n'a pas été lancé sur le jeu de données *C. elegans* au vu de ses temps d'exécution trop importants. [4] Daccord a reporté une erreur dès le lancement de l'étape d'alignement avec DALIGNER, sur le jeu de données *C. elegans*.

Ces résultats confirment les observations de la Section 7.3.1.1, toutes les méthodes d'auto-correction, y compris LoRMA, ayant permis de réduire les taux d'erreurs de tous les jeux de données en dessous de 1%. Concernant les méthodes de correction hybride, au vu des importants temps d'exécution de CoLoRMap, de HG-CoLoR et de Proovread sur le jeu de données *C. elegans* de la Table 7.4, disposant d'une profondeur de séquençage de 30x, ces derniers n'ont pas été lancés sur le jeu de données *C. elegans* étudié ici, et disposant d'une profondeur de séquençage de 60x. L'ensemble des méthodes de correction hybride ayant été lancées sur ce jeu de données, Jabba exclus, ont cependant produit des *reads* longs corrigés affichant des taux d'erreurs supérieurs à 1%, comme lors des expériences de la Section 7.3.1.1.

L'augmentation de la profondeur de séquençage n'a cependant pas permis aux méthodes d'auto-correction d'accroître significativement la qualité des *reads* longs corrigés, à part pour LoRMA. Au contraire, Canu a même produit des *reads* corrigés affichant des taux d'erreurs plus élevés à partir de ces jeux de données plus couverts. Les autres méthodes, bien qu'ayant effectivement bénéficié de cette augmentation de la profondeur de séquençage, n'ont permis de réduire que de 0,1339% en moyenne les taux d'erreurs des *reads* corrigés, la diminution la plus notable étant de 0,3616%, pour FLAS sur le jeu de données *C. elegans*.

Ces résultats confirment donc que l'auto-correction est parfaitement adaptée au traitement de données disposant de taux d'erreurs avoisinant les 12%, et ne nécessite pas d'importantes profondeurs de séquençage pour fournir des *reads* corrigés de haute qualité. En particulier, ces résultats soulignent même que l'augmentation de la profondeur de séquençage n'a que peu d'impact sur la qualité de la correction.

7.3.2 Données réelles

7.3.2.1 Qualité des *reads* longs corrigés

Outil	Métrique	Jeu de données	
		D. melanogaster	*H. sapiens*
CoLoRMap	Nombre de *reads*	564 080	419 332
	Nombre de bases	2 284 818 712	1 510 947 079
	Longueur moyenne (bp)	4 050	3 603
	Reads alignés (%)	99,8369	99,9764
	Identité moyenne (%)	96,6525	96,4502
	Couverture du génome (%)	97,5987	91,9475
	Temps	296 h 34 min	304 h 10 min
	Mémoire (Mo)	87 417	80 613
FMLRC	Nombre de *reads*	1 327 569	1 075 867
	Nombre de bases	9 190 808 479	7 647 637 034
	Longueur moyenne (bp)	6 923	7 108
	Reads alignés (%)	89,3343	93,1494
	Identité moyenne (%)	98,0212	95,3985
	Couverture du génome (%)	98,4638	92,4947
	Temps	88 h 11 min	77 h 15 min
	Mémoire (Mo)	10 111	16 381
HALC	Nombre de *reads*	1 288 954	1 065 270
	Nombre de bases	8 731 079 504	7 256 690 308
	Longueur moyenne (bp)	6 773	6 812
	Reads alignés (%)	97,0050	97,4287
	Identité moyenne (%)	98,0126	94,9728
	Couverture du génome (%)	98,4371	92,4495
	Temps	15 h 06 min	26 h 47 min
	Mémoire (Mo)	9 202	7 739
HG-CoLoR	Nombre de *reads*	1 139 148	970 212
	Nombre de bases	8 013 577 010	6 553 413 717
	Longueur moyenne (bp)	7 034	6 754
	Reads alignés (%)	99,7396	99,8157
	Identité moyenne (%)	98,8996	98,8042
	Couverture du génome (%)	98,5678	92,4523
	Temps	114 h 29 min	167 h 47 min
	Mémoire (Mo)	31 737	50 898
Jabba	Nombre de *reads*	1 527 456	1 059 671
	Nombre de bases	4 352 878 905	3 460 536 974
	Longueur moyenne (bp)	2 849	3 265
	Reads alignés (%)	89,5674	89,8926
	Identité moyenne (%)	99,9836	99,9888
	Couverture du génome (%)	92,7476	87,7037
	Temps	2 h 29 min	8 h 23 min
	Mémoire (Mo)	13 361	25 917
LoRDEC	Nombre de *reads*	1 310 770	1 075 867
	Nombre de bases	8 595 218 970	6 850 695 672
	Longueur moyenne (bp)	6 557	6 368
	Reads alignés (%)	87,7792	89,9923
	Identité moyenne (%)	96,7923	91,7205
	Couverture du génome (%)	98,4536	92,4693
	Temps	10 h 39 min	12 h 52 min
	Mémoire (Mo)	5 770	7 902

Outil	Métrique	Jeu de données	
		D. melanogaster	*H. sapiens*
Canu[1,2]	Nombre de *reads*	829 965	-
	Nombre de bases	6 992 565 713	-
	Longueur moyenne (bp)	8 425	-
	Reads alignés (%)	98,0543	-
	Identité moyenne (%)	95,2043	-
	Couverture du génome (%)	97,8902	-
	Temps	14 h 04 min	-
	Mémoire (Mo)	10 295	-
CONSENT	Nombre de *reads*	1 065 621	869 462
	Nombre de bases	8 178 293 508	6 348 740 755
	Longueur moyenne (bp)	7 674	7 301
	Reads alignés (%)	99,2593	99,5864
	Identité moyenne (%)	96,7179	93,0004
	Couverture du génome (%)	98,1981	92,3993
	Temps	38 h 00 min	8 h 30 min
	Mémoire (Mo)	51 361	45 869
Daccord[1,3]	Nombre de *reads*	-	-
	Nombre de bases	-	-
	Longueur moyenne (bp)	-	-
	Reads alignés (%)	-	-
	Identité moyenne (%)	-	-
	Couverture du génome (%)	-	-
	Temps	-	-
	Mémoire (Mo)	-	-
FLAS[1]	Nombre de *reads*	938 392	717 973
	Nombre de bases	7 866 243 928	5 695 206 077
	Longueur moyenne (bp)	8 382	7 932
	Reads alignés (%)	95,6517	99,0611
	Identité moyenne (%)	94,9925	90,9952
	Couverture du génome (%)	98,1012	92,3690
	Temps	10 h 18 min	4 h 57 min
	Mémoire (Mo)	18 820	14 957
LoRMA	Nombre de *reads*	13 920 047	6 796 439
	Nombre de bases	6 386 351 401	1 247 317 428
	Longueur moyenne (bp)	458	183
	Reads alignés (%)	97,0463	96,4997
	Identité moyenne (%)	98,4749	97,8254
	Couverture du génome (%)	94,7561	28,6189
	Temps	23 h 51 min	13 h 07 min
	Mémoire (Mo)	65 536	50 435
MECAT[1]	Nombre de *reads*	849 704	667 532
	Nombre de bases	7 287 782 016	5 479 325 177
	Longueur moyenne (bp)	8 576	8 208
	Reads alignés (%)	99,8681	99,9495
	Identité moyenne (%)	96,5229	91,6925
	Couverture du génome (%)	97,3432	91,4387
	Temps	1 h 54 min	1 h 53 min
	Mémoire (Mo)	13 443	11 075

TABLE 7.6 – **Comparaison de la qualité des *reads* longs corrigés par les différentes méthodes de l'état de l'art, sur les jeux de données réelles de la Table 6.3.** [1] Les *reads* plus longs que 50 000 paires de bases ont été filtrés du jeu de données *H. sapiens*, car les programmes s'arrêtaient en reportant une erreur lors de leur traitement. Le jeu de données initial contenait 1 824 de ces *ultra-long reads*, représentant un total de 135 364 312 paires de bases. [2] Canu s'est arrêté en reportant une erreur lors de la correction du jeu de données *H. sapiens*. [3] Daccord a reporté une erreur dès le lancement de l'étape d'alignement avec DALIGNER, sur les deux jeux de données.

Sur ces jeux de données plus bruitées, les méthodes de correction hybride ont de nouveau fourni de meilleurs résultats que les méthodes d'auto-correction. De plus, alors que seul CONSENT est parvenu à traiter les *ultra-long reads* contenus dans le jeu de données *H. sapiens*, toutes les méthodes de correction hybride ont permis leur traitement. Cependant, contrairement aux expériences de la Section 7.2.2.1, les méthodes d'auto-correction sont, ici, tout de même parvenues à grandement réduire les taux d'erreurs des *reads*. De plus, ces méthodes d'auto-correction ont permis de produire des *reads* corrigés affichant des longueurs moyennes supérieures aux *reads* corrigés produits par les méthodes de correction hybride, et couvrant des proportions comparables des génomes de référence. Par ailleurs, les méthodes d'auto-correction se sont également montrées globalement plus rapides que les méthodes de correction hybride, notamment FMLRC, HG-CoLoR, et plus particulièrement CoLoRMap.

L'avantage des méthodes de correction hybride sur les méthodes d'auto-correction, en termes de réduction des taux d'erreurs, est cependant cohérent avec les données étudiées. En effet, les jeux de données considérés ici sont composés de *reads* longs ONT. Ces *reads* affichent donc des taux d'erreurs supérieurs aux *reads* simulés PacBio étudiés dans la Section 7.3.1, mais souffrent également de biais et d'erreurs systématiques au sein des homopolymers. La correction de ces derniers est donc grandement facilitée par l'utilisation de *reads* courts complémentaires, expliquant ainsi les différences de qualité observées entre les *reads* corrigés par les méthodes de correction hybride et par les méthodes d'auto-correction.

Les méthodes d'auto-correction ont cependant permis d'atteindre un compromis satisfaisant entre qualité des résultats et consommation de ressources. En effet, ces méthodes ont permis de réduire à moins de 4% le taux d'erreurs des *reads* du jeu de données *D. melanogaster*, et à moins de 8% le taux d'erreurs des *reads* du jeu de données *H. sapiens*, affichant originalement des taux d'erreurs respectifs de 14,55% et 17,60%. Ces méthodes d'auto-correction ont également permis un gain de temps important par rapport à la majorité des méthodes de correction hybride.

7.3.2.2 Qualité des assemblages

Outil	Métrique	Jeu de données	
		D. melanogaster	*H. sapiens*
CoLoRMap	Nombre de contigs	1 388	404
	Nombre de contigs alignés	1 388	404
	Nombre de *breakpoints*	337	52
	NGA50 (bp)	56 098	1 126
	NGA75 (bp)	2 521	1 126
	Couverture du génome (%)	67,5209	4,5428
FMLRC	Nombre de contigs	240	66
	Nombre de contigs alignés	238	63
	Nombre de *breakpoints*	1 440	275
	NGA50 (bp)	5 035 355	2 010
	NGA75 (bp)	1 009 708	2 010
	Couverture du génome (%)	91,9722	47,8800
HALC	Nombre de contigs	354	72
	Nombre de contigs alignés	351	69
	Nombre de *breakpoints*	1 917	256
	NGA50 (bp)	1 814 344	1 239
	NGA75 (bp)	743 496	1 239
	Couverture du génome (%)	91,7812	48,0811

Outil	Métrique	Jeu de données *D. melanogaster*	Jeu de données *H. sapiens*
HG-CoLoR	Nombre de contigs	262	196
	Nombre de contigs alignés	262	196
	Nombre de *breakpoints*	1 137	393
	NGA50 (bp)	4 445 613	4 628
	NGA75 (bp)	1 328 146	4 628
	Couverture du génome (%)	91,1995	49,6926
Jabba	Nombre de contigs	816	2 032
	Nombre de contigs alignés	816	2 032
	Nombre de *breakpoints*	32	83
	NGA50 (bp)	111 022	3 418
	NGA75 (bp)	6 007	3 418
	Couverture du génome (%)	70,3394	28,8106
LoRDEC	Nombre de contigs	310	111
	Nombre de contigs alignés	309	108
	Nombre de *breakpoints*	1 813	655
	NGA50 (bp)	4 846 371	2 281
	NGA75 (bp)	1 283 448	2 281
	Couverture du génome (%)	91,581	48,3585
Canu[1,2]	Nombre de contigs	276	-
	Nombre de contigs alignés	269	-
	Nombre de *breakpoints*	1 269	-
	NGA50 (bp)	4 517 869	-
	NGA75 (bp)	1 293 578	-
	Couverture du génome (%)	92,1034	-
CONSENT	Nombre de contigs	321	248
	Nombre de contigs alignés	317	245
	Nombre de *breakpoints*	2 508	871
	NGA50 (bp)	8 077 282	3 760 455
	NGA75 (bp)	2 009 827	1 368 818
	Couverture du génome (%)	92,5688	88,9587
Daccord[1,3]	Nombre de contigs	-	-
	Nombre de contigs alignés	-	-
	Nombre de *breakpoints*	-	
	NGA50 (bp)	-	-
	NGA75 (bp)	-	-
	Couverture du génome (%)	-	-
FLAS[1]	Nombre de contigs	265	259
	Nombre de contigs alignés	264	258
	Nombre de *breakpoints*	1 379	317
	NGA50 (bp)	4 220 587	1 622 624
	NGA75 (bp)	830 179	346 914
	Couverture du génome (%)	92,1105	88,4068
LoRMA	Nombre de contigs	-	4
	Nombre de contigs alignés	-	4
	Nombre de *breakpoints*	-	0
	NGA50 (bp)	-	2 742
	NGA75 (bp)	-	2 742
	Couverture du génome (%)	-	0,0060
MECAT[1]	Nombre de contigs	370	249
	Nombre de contigs alignés	370	248
	Nombre de *breakpoints*	1 058	199
	NGA50 (bp)	853 544	1 783 022
	NGA75 (bp)	278 555	462 658
	Couverture du génome (%)	89,5839	88,7002

TABLE 7.7 – **Comparaison de la qualité des assemblages générés à partir des *reads* longs corrigés par les différentes méthodes de l'état de l'art, sur les jeux de données réelles de la Table 6.3.** [1] Les *reads* plus longs que 50 000 paires de bases ont été filtrés du jeu de données *H. sapiens*, car les programmes s'arrêtaient en reportant une erreur lors de leur traitement. Le jeu de données initial contenait 1 824 de ces *ultra-long reads*, représentant un total de 135 364 312 paires de bases. [2] Canu s'est arrêté en reportant une erreur lors de la correction du jeu de données *H. sapiens*, malgré la filtration des *ultra-long reads*. [3] Daccord a reporté une erreur dès le lancement de l'étape d'alignement avec DALIGNER, sur les deux jeux de données.

Malgré leurs taux d'erreurs plus élevés, les *reads* traités par les méthodes d'auto-correction, LoRMA exclus, ont permis de générer des assemblages de meilleure qualité que les *reads* traités par les méthodes de correction hybride. En particulier, sur le jeu de données *D. melanogaster*, les méthodes d'auto-correction, MECAT exclus, ont produit des *reads* corrigés permettant de générer des assemblages couvrant de plus larges proportions du génome de référence, et affichant des tailles NGA50 et NGA75 globalement plus élevées. Cette différence de qualité s'est creusée davantage sur le jeu de données *H. sapiens*. Les *reads* traités par les méthodes d'auto-correction ont, en effet, permis de générer des assemblages couvrant une moyenne de plus de 88% du génome de référence. Les *reads* traités par les méthodes de correction hybride, quant à eux, ont permis de générer des assemblages ne couvrant qu'une moyenne de 50% du génome de référence. L'assemblage généré à partir des *reads* corrigés par CoLoRMap n'a même permis de couvrir que 4,5428% du génome.

Les tailles NGA50 et NGA75 des assemblages du jeu de données *H. sapiens* ont également souligné l'avantage des méthodes d'auto-correction. En effet, les tailles NGA50 et NGA75 des assemblages générés à partir des *reads* traités par les méthodes de correction hybride n'ont atteint que quelques milliers de paires bases. Les tailles NGA50 et NGA75 des assemblages générés à partir des *reads* traités par les méthodes d'auto-correction, en revanche, ont atteint respectivement quelques millions et plusieurs centaines de milliers de paires de bases. Comme mentionné Section 7.3.1.1, ces résultats sont cohérents avec les spécificités de l'auto-correction, permettant de résoudre plus aisément les répétitions, en tirant pleinement profit de l'information portée par la longueur des *reads*.

Ces résultats soulignent ainsi que, pour des jeux de données provenant de génomes complexes disposant de nombreuses répétitions, et affichant des taux d'erreurs relativement élevés, de l'ordre de 18%, l'auto-correction est plus adaptée que la correction hybride pour le traitement de *reads* en vu d'un assemblage.

Chapitre 8

Conclusion et perspectives

Sommaire

Ce chapitre présente la conclusion générale de cette thèse, ainsi que ses perspectives, proposées aussi bien à l'échelle des différents résultats décrits, que de manière plus générale.

8.1 Conclusion

Les objectifs de cette thèse s'inscrivent ainsi dans la large problématique de l'analyse des données de séquençage, et plus particulièrement des *reads* longs issus des technologies de séquençage de troisième génération. Les aspects abordés dans cette problématique se sont principalement concentrés sur la correction des erreurs de séquençage, présentes en grande proportion dans ces *reads*. D'une part, la correction hybride, utilisant en complément des *reads* courts et précis, issus de technologies de séquençage de deuxième génération, ainsi que l'auto-correction, utilisant uniquement l'information contenue au sein des séquences des *reads* longs, ont été étudiées. D'autre part, l'impact de cette étape de correction sur les résultats d'analyses, et plus

particulièrement sur la qualité des assemblages pouvant être générés, a également été analysé.

Comme présenté au sein du Chapitre 3, une large variété d'outils, aussi bien de correction hybride que d'auto-correction, existe et repose sur des méthodologies variées. Quatre approches ont ainsi été décrites pour la correction hybride : l'alignement de *reads* courts sur les *reads* longs, l'alignement de *reads* longs et d'assemblages de *reads* courts, l'utilisation de graphes de de Bruijn, et l'utilisation de modèles de Markov cachés. Concernant l'auto-correction, deux approches ont été décrites : l'alignement multiple, et l'utilisation de graphes de de Bruijn. Comme souligné par les diverses expériences réalisées dans le Chapitre 7, différents paramètres, tels que la profondeur de séquençage, la longueur et le taux d'erreurs des *reads*, ou encore la complexité du génome étudié, peuvent impacter l'efficacité de ces différentes méthodes.

Les travaux menés dans le cadre de cette thèse se sont donc décomposés en trois principaux objectifs. Tout d'abord, au vu de l'impact des propriétés des jeux de données sur l'efficacité des différentes méthodes de correction, un premier objectif rapidement identifié fut la nécessité de développer un outil permettant d'évaluer de façon précise et détaillée la qualité de la correction fournie par une méthode donnée. À l'aide de cet outil, des analyses ont ensuite été réalisées sur les différentes méthodes de correction de l'état de l'art, et ont permis d'identifier deux limitations majeures. Dans un premier temps, ces analyses ont souligné la difficulté des outils de l'état de l'art à corriger des *reads* affichant des taux d'erreurs supérieurs à 30%. Le second objectif de cette thèse fut donc le développement d'un outil de correction hybride, destiné au traitement de ces *reads* extrêmement bruités. Dans un second temps, ces analyses ont également souligné que les outils d'auto-correction de l'état de l'art sont limités par la longueur des *reads*, et ne peuvent notamment pas être appliqués aux librairies *ultra-long reads* ONT, pouvant produire des *reads* atteignant jusqu'à un million de paires de bases. Le troisième objectif de cette thèse fut alors le développement d'un outil d'auto-correction s'affranchissant des limitations liées à la longueur des *reads*, et permettant ainsi la correction de données *ultra-long reads*.

8.1.1 Contributions

8.1.1.1 ELECTOR

L'un des premiers objectifs de cette thèse, abordé au sein du Chapitre 4, fut le développement d'un outil permettant d'évaluer de manière précise et détaillée la qualité de la correction fournie par les différents outils disponibles. Un tel outil offre en effet la possibilité de réaliser des *benchmarks* de grande envergure, incluant de nombreux outils, et des jeux de données variés. Réaliser de tels *benchmarks* présente deux principaux intérêts. D'une part, ils permettent aux biologistes d'identifier rapidement et aisément quelle méthode de correction est la plus adaptée à leur jeu de données. D'autre part, ils permettent également aux développeurs de comparer précisément leurs outils aux autres outils de l'état de l'art, et peuvent ainsi être utilisés dans le cadre de publications. Dans ce contexte, nous avons développé ELECTOR, un outil d'évaluation de la qualité de la correction. ELECTOR propose de comparer trois versions de chaque *read* : la version non corrigée, la version corrigée, et la version de référence, représentant la région du génome de référence dont a été extrait le *read*. Un alignement multiple est ainsi calculé à partir de ces trois séquences, afin d'identifier précisément, à chaque position, les similarités et les différences entre les

trois différentes versions du *read*. De nombreuses métriques peuvent ainsi être calculées, dont notamment les taux d'erreurs des *reads* et les comptes précis des différents types d'erreurs (substitutions, insertions et délétions) avant et après correction, mais également le rappel, indiquant la capacité de l'outil à identifier et à corriger les erreurs, la précision, indiquant la capacité de l'outil à ne pas introduire d'erreurs supplémentaires lors de la correction, ainsi que le nombre de *reads trimmés*, *splittés*, ou étendus par la correction. Comparé à LRCstats, le seul outil précédemment disponible pour évaluer la qualité de la correction, et ne fournissant que les taux d'erreurs des *reads* et les comptes des différents types d'erreurs, avant et après correction, ELECTOR permet donc une évaluation bien plus précise. De plus, afin de permettre à la stratégie d'alignement multiple de passer à l'échelle, aussi bien sur des *reads* extrêmement longs que sur des génomes de taille importante, une stratégie de segmentation a été développée. Cette stratégie permet de diviser le problème de l'alignement multiple en plusieurs instances de plus petite taille, alors moins coûteuses à calculer. En particulier, grâce à cette stratégie de segmentation, ELECTOR s'est montré jusqu'à 22 fois plus rapide que LRCstats sur l'ensemble des jeux de données évalués, attestant ainsi de sa meilleure capacité de passage à l'échelle. Cette capacité de passage à l'échelle a, de plus, été validée davantage, notamment par des expériences sur des données humaines.

8.1.1.2 HG-CoLoR

Le deuxième objectif de cette thèse, décrit au sein du Chapitre 5, fut le développement d'un outil de correction hybride, destiné à la correction des *reads* longs affichant des taux d'erreurs supérieurs à 30%. En effet, bien que les taux d'erreurs des *reads* longs soient compris entre 10 et 30% en moyenne, les premières expériences de séquençage pouvaient produire des *reads* extrêmement bruités, affichant des taux d'erreurs supérieurs à 30%. Malgré les récentes avancées des technologies de séquençage de troisième génération, permettant désormais d'atteindre des taux d'erreurs de 10 à 12% en moyenne, permettre la correction de *reads* fortement bruités présente tout de même deux intérêts. Tout d'abord, corriger des données affichant de tels taux d'erreurs afin de les amener à un haut niveau de qualité représente un important challenge algorithmique, quelle que soit l'approche de correction utilisée. De plus, dès leur apparition, les technologies de séquençage de troisième génération ont permis de générer d'importantes quantités de données. Le reséquençage étant coûteux, il n'est pas accessible à l'intégralité des laboratoires. Corriger ces données représente donc un intérêt majeur pour ces derniers, afin de leur permettre de réaliser des analyses efficaces. Dans cette optique, nous avons développé HG-CoLoR, un outil de correction hybride destiné aux *reads* affichant d'importants taux d'erreurs. HG-CoLoR combine différentes approches de l'état de l'art de la correction hybride, afin de palier les limitations inhérentes à chacune d'elle. Ainsi, les *reads* courts sont tout d'abord alignés sur les *reads* longs, afin d'initier le processus de correction. Les *reads* courts alignés sont ensuite utilisés comme ancres sur un graphe de de Bruijn, au sein duquel des chemins permettant de relier ces ancres sont recherchés. Ces chemins sont alors utilisés afin de corriger les régions non couvertes des *reads* longs. De plus, ce graphe de de Bruijn a la particularité d'être d'ordre variable, et permet ainsi d'éviter les biais liés au choix de l'ordre au moment de la construction. HG-CoLoR permet ainsi la correction de *reads* longs extrêmement bruités, réduisant notamment de plus de 44% à moins de 0,3% le taux d'erreurs d'un jeu de données ONT séquencé avec une des premières versions des chimies de cette technologie. Comparé

aux autres méthodes de l'état de l'art, HG-CoLoR fournit ainsi un excellent compromis entre qualité de la correction et consommation de ressources, et permet également un meilleur passage à l'échelle, notamment sur des génomes eucaryotes. De plus, nos expériences ont également montré que la qualité de la correction fournie par HG-CoLoR permet d'obtenir des assemblages de haute qualité, surpassant les assemblages obtenus à partir des *reads* corrigés par les autres méthodes de l'état de l'art.

8.1.1.3 CONSENT

Le troisième objectif de cette thèse, présenté au sein du Chapitre 6, fut le développement d'un outil d'auto-correction, destiné à la correction des *reads* extrêmement longs. En effet, l'évolution des technologies de séquençage de troisième génération ne permet pas uniquement la diminution des taux d'erreurs des *reads*, mais permet également de séquencer des *reads* de plus en plus longs. En particulier, la longueur des *reads* ONT n'est pas limitée par la plateforme de séquençage utilisée, mais uniquement par la préparation de la librairie utilisée pour le séquençage. Ainsi, des librairies permettant le séquençage de *reads* de plus d'un million de paires de bases, appelés *ultra-long reads*, ont récemment été développées. Bien que les méthodes d'auto-correction puissent efficacement être utilisées sur les *reads* longs issus d'expériences de séquençage récentes, grâce à la diminution des taux d'erreurs mentionnée précédemment, ces méthodes ne permettent pas de passer à l'échelle sur les librairies *ultra-long reads*. Au vu de l'intérêt de ces *reads* pour la résolution de problèmes d'assemblage de génomes longs et complexes, contenant de longues répétitions, nous avons développé CONSENT, un outil d'auto-correction destiné à leur traitement. CONSENT combine ainsi différentes approches de l'état de l'art, en commençant par calculer les chevauchements entre les *reads* en utilisant une approche par *mapping*. Les *reads* sont alors corrigés à l'aide des autres *reads* avec lesquels ils partagent des chevauchements. Les chevauchements d'un *read* donné sont ainsi divisés en fenêtres de petite taille, et une approche par alignement multiple est ensuite utilisée afin de corriger chacune de ces fenêtres. Afin d'assurer une bonne capacité de passage à l'échelle, et un calcul rapide des alignements multiples, la stratégie de segmentation introduite pour ELECTOR, dans le Chapitre 4, a ici été généralisée. Une fois cette étape de correction par alignement multiple réalisée, la correction est alors raffinée en utilisant des graphes de de Bruijn locaux, à l'échelle des fenêtres, afin de réduire davantage les taux d'erreurs. CONSENT s'est ainsi montré qualitativement comparable aux autres méthodes d'auto-correction de l'état de l'art, aussi bien du point de vue de la correction en elle-même que du point de vue des assemblages pouvant être générés à partir des *reads* corrigés. De plus, CONSENT s'est révélé être le seul outil d'auto-correction capable de passer à l'échelle sur un jeu de données du génome humain, contenant des *ultra-long reads* ONT atteignant jusqu'à 340 000 paires de bases. Une fonctionnalité additionnelle de correction d'assemblages, non disponible dans les autres outils de correction, a également été implémentée. Cette fonctionnalité a permis d'obtenir des résultats satisfaisants, mais également un important gain de temps par rapport à un des outils de correction d'assemblages les plus largement utilisés.

8.1.1.4 *Benchmark* comparatif des méthodes de correction

Dans le Chapitre 7, nous avons proposé un *benchmark* complet de l'ensemble des méthodes de correction disponibles, sur des jeux de données variés, de la bactérie

à l'humain. La longueur et le taux d'erreurs des *reads*, ainsi que la profondeur de séquençage de ces jeux de données, ont été choisis afin de représenter différents facteurs limitants, et d'étudier en détail le comportement des différentes méthodes de correction.

Ce *benchmark* a notamment montré que, même avec d'importantes profondeurs de séquençage, les méthodes d'auto-correction ne sont pas adaptées au traitement de taux d'erreurs supérieurs à 20%, typiques des *reads* ONT issus des premières expériences de séquençage. Ainsi, pour de tels *reads*, la correction hybride demeure l'alternative la plus efficace, bien que globalement plus coûteuse en temps.

Au contraire, pour le traitement de *reads* affichant des taux d'erreurs plus faibles, les méthodes d'auto-correction se sont montrées comparables, voire même plus efficaces que les méthodes de correction hybride, tout en permettant une importante diminution du temps d'exécution. Sur des organismes complexes tels que celui de l'humain, la qualité des assemblages obtenus après traitement des *reads* par auto-correction s'est notamment révélée supérieure à la qualité des assemblages obtenus après traitement des *reads* par correction hybride, particulièrement en termes de couverture du génome. Ces observations sont cependant cohérentes avec les limitations inhérentes à la correction hybride, le calcul de chevauchements entre les *reads* longs eux-mêmes permettant en effet une identification et une correction plus aisées des régions répétées.

Ainsi, au vu des taux d'erreurs affichés par les *reads* longs issus d'expériences de séquençage récentes, l'auto-correction semble actuellement s'imposer comme l'alternative la plus efficace.

8.2 Perspectives

8.2.1 ELECTOR

8.2.1.1 Traitement parallèle des alignements multiples

Bien qu'ELECTOR permette une grande réduction du temps de traitement par rapport à LRCstats, l'évaluation de la correction du jeu de données *C. elegans* par HALC, présentée Table 4.6, et l'évaluation de la correction du jeu de données *H. sapiens* par Canu, présentée Table 4.11, ont tout de même affiché des temps d'exécution importants.

Ces derniers ne sont cependant pas liés aux calculs des alignements multiples, mais à l'étape d'analyse de ces alignements, permettant de calculer l'ensemble des métriques décrivant la qualité de la correction. En effet, lors de cette étape, l'ensemble des alignements multiples est écrit au sein d'un fichier unique. Le parcours de ce fichier, et l'analyse des alignements multiples qu'il contient, ne sont alors pas réalisés en parallèle par ELECTOR. Un tel comportement peut donc se révéler particulièrement lent, notamment dans les cas où la méthode de correction évaluée produit de nombreux *reads splittés*, comme HALC sur le jeu de données *C. elegans* de la Table 4.6. En effet, ces *reads splittés* impliquent un plus grand nombre d'alignements multiples à analyser, et affectent donc le temps d'exécution. Traiter le fichier contenant ces alignements multiples en parallèle, en le divisant par exemple en plusieurs fichiers de plus petite taille, pourrait ainsi permettre une importante réduction du temps d'exécution.

8.2.1.2 Compléments au module d'évaluation des assemblages

Le module d'évaluation des assemblages d'ELECTOR ne permet, à l'heure actuelle, de réaliser les assemblages qu'à l'aide de Miniasm. Comme mentionné Section 4.4.2, bien que cet outil soit rapide, d'autres assembleurs tels que Canu, Smartdenovo, ou FALCON sont plus précis, et permettent donc d'obtenir des assemblages de meilleure qualité. Ces assembleurs pourraient donc être intégrés au module d'évaluation d'ELECTOR, en laissant à l'utilisateur le choix de l'assembleur à utiliser, en fonction du compromis désiré entre précision et temps d'exécution.

De plus, seules des métriques décrivant la qualité de l'assemblage de manière globale sont reportées par ELECTOR. Comparé à QUAST-LG, dont une partie des métriques a été décrite lors de l'expérience présentée Table 6.11, ELECTOR est donc bien moins indicatif. En particulier, le taux d'identité entre l'assemblage évalué et le génome de référence n'est pas reporté par ELECTOR, et le taux d'erreurs de l'assemblage ne peut ainsi pas être connu. Au vu du faible temps d'exécution et de la faible consommation de ressources de QUAST-LG, celui-ci pourrait également être intégré à ELECTOR, en laissant de nouveau le choix à l'utilisateur concernant son exécution.

8.2.2 HG-CoLoR

8.2.2.1 Remplacement de BLASR

L'étape d'alignement des *reads* courts sur les *reads* longs est actuellement réalisée à l'aide de BLASR. Bien que permettant de mettre en évidence des graines de haute qualité, pouvant être efficacement utilisées comme ancres sur le graphe, BLASR souffre de temps d'exécution importants, ralentissant le processus de correction de HG-CoLoR. BLASR pourrait donc être remplacé par Minimap2, ayant été adopté comme son remplaçant officiel par la communauté [1] [2]. De plus, Minimap2 dispose d'un mode *mapping* et d'un mode alignement. Il serait donc intéressant d'étudier l'impact du choix du mode sur la qualité des résultats, et le compromis entre temps d'exécution et qualité des résultats pouvant être obtenu, en fonction du mode choisi. Une comparaison des temps d'exécution de BLASR, et des modes *mapping* et alignement de Minimap2, sur les jeux de données réelles utilisés dans le Chapitre 5, est présentée Table 8.1 Cette comparaison souligne le net avantage de Minimap2 sur BLASR, quel que soit le mode utilisé.

Outil	Jeu de données			
	A. baylyi	*E. coli*	*S. cerevisiae*	*C. elegans*
BLASR	1 h 05 min	48 min	2 h 37 min	24 h 41 min
Minimap2 (*mapping*)	42 sec	30 sec	2 min	10 min
Minimap2 (alignement)	8 min	7 min	16 min	1 h 42 min

TABLE 8.1 – **Comparaison des temps d'exécution de BLASR et de Minimap2.** Les jeux de données correspondent aux jeux de données réelles de la Table 5.3.

1. https://twitter.com/lh3lh3/status/1085935299031650304
2. https://github.com/PacificBiosciences/pbmm2

8.2.2.2 Remplacement de PgSA

Un autre facteur limitant, affectant le temps d'exécution de HG-CoLoR, est l'utilisation de PgSA pour la représentation du graphe de de Bruijn d'ordre variable. En effet, PgSA ne permet pas d'interroger l'index en parallèle, et nécessite donc la mise en place de verrous d'exclusion mutuelle, afin de s'assurer de son bon fonctionnement. Au vu du grand nombre de traversées du graphe réalisées durant le processus de correction, cette impossibilité d'interroger l'index en parallèle ralentit donc grandement HG-CoLoR. PgSA pourrait ainsi être remplacé par une autre structure d'indexation, permettant de représenter un graphe de de Bruijn d'ordre variable, et pouvant être interrogée en parallèle. Le FM-index, utilisé au sein de FMLRC, pourrait par exemple être adopté. Le remplacement de BLASR et de PgSA pourrait ainsi grandement diminuer les temps d'exécution de HG-CoLoR, et lui permettre de passer plus efficacement à l'échelle sur de larges génomes.

8.2.3 CONSENT

8.2.3.1 Optimisation des paramètres

Comme décrit tout au long du Chapitre 6, CONSENT dispose de nombreux paramètres pouvant affecter la qualité de la correction, et pour lesquels des valeurs par défaut ont été proposées. Cependant, contrairement à HG-CoLoR, pour lequel une étude de l'impact des paramètres sur les résultats a été réalisée Section 5.4.1, une telle étude n'a pas été menée pour CONSENT. Ainsi, réaliser une étude détaillée de l'impact des paramètres pourrait permettre de mettre en valeur un nouvel ensemble de valeurs à utiliser, et pourrait également permettre d'améliorer la qualité de la correction fournie par CONSENT. De plus, comme le soulignent les expériences réalisées Section 6.3.3, la qualité de la correction réalisée par l'étape d'alignement multiple impacte directement le coût de l'étape de raffinement avec les graphes de de Bruijn. Ainsi, adapter davantage les paramètres pourrait permettre à l'étape de correction par alignement multiple de produire des résultats de plus haute qualité, et donc de réduire le coût de l'étape de raffinement.

8.2.3.2 Filtrage des chevauchements

Les expériences réalisées Section 6.3.5 ont montré que le temps d'exécution de CONSENT, et plus particulièrement de l'étape de correction, tend à augmenter en fonction de la complexité du génome. Ce comportement peut être expliqué par la présence de répétitions plus nombreuses au sein des génomes plus complexes. Ces répétitions impactent alors la couverture des piles de chevauchements, et peuvent ainsi mener au traitement de fenêtres affichant des couvertures extrêmement importantes. De telles fenêtres contiennent alors un grand nombre de séquences à traiter et à aligner, et impactent ainsi le temps d'exécution, malgré l'utilisation de notre stratégie de segmentation. Ainsi, une stratégie additionnelle de validation, permettant de considérer uniquement les *reads* provenant de la même copie de la répétition, au sein d'une pile de chevauchements, pourrait être employée. Une telle stratégie permettrait en effet de réduire la couverture moyenne des piles et des fenêtres, résultant ainsi en un plus faible nombre de séquences à traiter lors de l'étape d'alignement multiple, et donc en une réduction du temps d'exécution. De plus, réduire le contenu des piles de chevauchements aux *reads* provenant des mêmes copies des répétitions pourrait également avoir un impact positif sur la qualité de la correction.

8.2.3.3 Optimisation de la parallélisation

CONSENT est actuellement parallélisé de façon à traiter plusieurs séquences en parallèle, et non plusieurs fenêtres. Les contigs étant longs et peu nombreux par rapport aux *reads*, cette méthode de parallélisation n'est donc pas idéalement adaptée à la problématique de la correction d'assemblages. En particulier, dans le cas des bactéries et des archées, ne possédant généralement qu'un seul chromosome, CONSENT ne sera pas en mesure de paralléliser le processus de correction. En effet, ces assemblages, s'ils sont de bonne qualité, seront composés d'un unique contig, et CONSENT traitera alors séquentiellement les différentes fenêtres issues des chevauchements entre ce contig et les *reads* longs utilisés pour sa correction. Ainsi, modifier le processus de parallélisation afin de permettre à CONSENT de traiter plusieurs fenêtres en parallèle, plutôt que plusieurs séquences, permettrait une importante réduction du temps d'exécution lors de la correction d'assemblages, sans pour autant avoir d'impact sur le processus de correction de *reads*.

8.2.4 Perspectives générales

De manière plus générale, le travail réalisé dans le cadre de cette thèse s'est uniquement focalisé sur la correction de données génomiques. Le problème de la correction de données transcriptomiques n'a donc pas été abordé. Des travaux récents, tels que [74], soulignent que les outils de correction de *reads* ADN sont également applicables à des données ARN, et permettent effectivement de réduire leurs taux d'erreurs. Cependant, ces travaux soulignent également que ces outils tendent à réduire le nombre de gènes détectés, et à corriger les différentes isoformes (*i.e.* les différentes formes que peut prendre une protéine) vers l'isoforme la plus représentée au sein des données brutes. Étudier ce comportement plus en détail, et développer de nouvelles méthodes de correction, davantage adaptées aux données transcriptomiques, et permettant de limiter les biais préalablement mentionnés, représenterait donc une continuité logique des travaux menés dans le cadre de cette thèse.

Dans un second temps, les expériences de correction d'assemblages, présentées Section 6.3.6, ont souligné le fait que d'assembler les *reads* bruts, puis corriger l'assemblage ainsi obtenu, était qualitativement comparable au fait de corriger les *reads*, puis de générer l'assemblage à partir des *reads* corrigés. En particulier, sur le jeu de données *H. sapiens*, assembler les *reads* puis corriger l'assemblage a même permis d'obtenir moins de contigs, et de couvrir une plus large proportion du génome, au prix d'une diminution des tailles NGA50 et NGA75. Ces observations sont notamment liées à l'évolution des technologies de séquençage de troisième génération, permettant maintenant de séquencer des *reads* plus longs et plus précis. Elles soulèvent cependant la question de l'intérêt de la correction des *reads* avant l'étape d'assemblage, en particulier au vu du coût non négligeable de la correction.

Étudier l'impact de l'ordre correction / assemblage ou assemblage / correction sur la qualité des résultats, à plus grande échelle, pourrait ainsi guider les futures recherches à emprunter un axe différent. Les problématiques de la correction de *reads* et de la correction d'assemblages sont cependant extrêmement proches, et reposent sur les mêmes principes algorithmiques. L'adaptation d'un problème à l'autre est alors aisée, comme l'a souligné la fonctionnalité de correction d'assemblages de CONSENT. Ainsi, l'une des principales perspectives de cette thèse est d'étudier plus en détail la problématique de la correction d'assemblages. L'optimisation de CONSENT pour cette problématique pourrait ainsi être un premier pas intéressant dans cette direction.

Liste des publications et communications

Publications dans des revues internationales avec comité de lecture

[1] Pierre MORISSE, Thierry LECROQ et Arnaud LEFEBVRE. Hybrid correction of highly noisy long reads using a variable-order de Bruijn graph. *Bioinformatics* 34.24 (2018), p. 4213-4222. DOI : 10.1093/bioinformatics/bty521.

Communications dans des conférences internationales avec comité de lecture

[2] Pierre MORISSE, Camille MARCHET, Antoine LIMASSET, Thierry LECROQ et Arnaud LEFEBVRE. CONSENT : Scalable self-correction of long reads with multiple sequence alignment. *RECOMB-SEQ,* Washington D.C, États-Unis d'Amérique (mai 2019).

Communications dans des conférences nationales avec comité de lecture

[3] Pierre MORISSE, Thierry LECROQ et Arnaud LEFEBVRE. HG-CoLoR : enHanced de Bruijn Graph for the error Correction of Long Reads *Seqbio 2017*, Lille, France, novembre 2017. *Journées Ouvertes en Biologie, Informatique et Mathématiques (JOBIM),* Lille, France (juillet 2017).

[4] Pierre MORISSE, Camille MARCHET, Antoine LIMASSET, Thierry LECROQ et Arnaud LEFEBVRE. CONSENT : Scalable self-correction of long reads with multiple sequence alignment. *Journées Ouvertes en Biologie, Informatique et Mathématiques (JOBIM),* Nantes, France (juillet 2019).

Communications dans des workshops internationaux

[5] Pierre MORISSE, Thierry LECROQ et Arnaud LEFEBVRE. Enhanced de Bruijn Graphs. *Mathematic foundations in Bioinformatics (MatBio),* Londres, Royaume-Uni (septembre 2017).

[6] Pierre MORISSE, Thierry LECROQ et Arnaud LEFEBVRE. Hybrid correction of long reads using a variable-order de Bruijn graph. *Data Structures in Bioinformatics (DSB),* Helsinki, Finlande (mai 2018).

[7] Pierre MORISSE, Camille MARCHET, Antoine LIMASSET, Thierry LECROQ et Arnaud LEFEBVRE. CONSENT : Scalable self-correction of long reads with multiple sequence alignment. *Data Structures in Bioinformatics (DSB),* Dortmund, Allemagne (février 2019).

Communications dans des workshops nationaux

[8] Camille MARCHET, Pierre MORISSE, Lolita LECOMPTE, Antoine LIMASSET, Arnaud LEFEBVRE et al. ELECTOR : EvaLuator of Error Correction Tools for lOng Reads. *SeqBio,* Rouen, France (novembre 2018).

[9] Pierre MORISSE, Thierry LECROQ et Arnaud LEFEBVRE. HG-CoLoR : enHanced de Bruijn Graph for the error Correction of Long Reads. *SeqBio,* Lille, France (novembre 2017).

[10] Pierre MORISSE, Antoine LIMASSET, Camille MARCHET, Arnaud LEFEBVRE, Pierre PETERLONGO et al. LoRSCo : Long Reads Self-Correction. *SeqBio,* Rouen, France (novembre 2018).

Conférences et séminaires invité

[11] Pierre MORISSE. Correction de données de séquençage de troisième génération. *Séminaire d'informatique théorique,* Rouen, France (mars 2019).

[12] Pierre MORISSE. Diverses approches pour l'auto-correction des lectures longues. *Séminaire Symbiose de l'Inria,* Rennes, France (octobre 2017).

Communications sous forme de posters

[13] Camille MARCHET, Pierre MORISSE, Lolita LECOMPTE, Antoine LIMASSET, Arnaud LEFEBVRE et al. ELECTOR : EvaLuation of Error Correction Tools for lOng Reads. *Journées Ouvertes en Biologie, Informatique et Mathématiques (JOBIM),* Marseille, France (juillet 2018).

Prépublications

[14] Camille MARCHET, Pierre MORISSE, Lolita LECOMPTE, Antoine LIMASSET, Arnaud LEFEBVRE et al. ELECTOR : Evaluator for long reads correction methods. *bioRxiv* (2019). DOI : 10.1101/512889.

[15] Pierre MORISSE, Camille MARCHET, Antoine LIMASSET, Thierry LECROQ et Arnaud LEFEBVRE. CONSENT : Scalable self-correction of long reads with multiple sequence alignment. *bioRxiv* (2019). DOI : 10.1101/546630.

Bibliographie

[1] Amin ALLAM, Panos KALNIS et Victor SOLOVYEV. Karect : accurate correction of substitution, insertion and deletion errors for next-generation sequencing data. *Bioinformatics* 31 (2015), p. 3421-3428. DOI : 10.1093/bioinformatics/btv415.

[2] S F ALTSCHUL, W GISH, W MILLER, E W MYERS et D J LIPMAN. Basic local alignment search tool. *Journal of Molecular biology* 215.3 (1990), p. 403-10. DOI : 10.1016/S0022-2836(05)80360-2.

[3] Eric L. ANSON et Eugene W. MYERS. ReAligner : A Program for Refining DNA Sequence Multi-Alignments. *Journal of Computational Biology* 4.3 (2009), p. 369-383. DOI : 10.1089/cmb.1997.4.369.

[4] Kin Fai AU, Jason G. UNDERWOOD, Lawrence LEE et Wing Hung WONG. Improving PacBio Long Read Accuracy by Short Read Alignment. *PLoS ONE* 7.10 (2012), p. 1-8. DOI : 10.1371/journal.pone.0046679.

[5] Anton BANKEVICH, Sergey NURK, Dmitry ANTIPOV, Alexey A GUREVICH, Mikhail DVORKIN et al. SPAdes : A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology* 19.5 (2012), p. 455-477. DOI : 10.1089/cmb.2012.0021.

[6] Ergude BAO et Lingxiao LAN. HALC : High throughput algorithm for long read error correction. *BMC Bioinformatics* 18 (2017), p. 204. DOI : 10.1186/s12859-017-1610-3.

[7] Ergude BAO, Fei XIE, Changjin SONG et Dandan SONG. FLAS : fast and high-throughput algorithm for PacBio long-read self-correction. *Bioinformatics* (2019). DOI : 10.1093/bioinformatics/btz206.

[8] Leonard E BAUM. An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. *Inequalities III : Proceedings of the Third Symposium on Inequalities*. Academic Press, 1972, p. 1-8.

[9] Konstantin BERLIN, Sergey KOREN, Chen-Shan CHIN, James P DRAKE, Jane M LANDOLIN et al. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotechnology* 33 (2015), p. 623-630. DOI : 10.1038/nbt.3238.

[10] Christina BOUCHER, Alex BOWE, Travis GAGIE, Simon J PUGLISI et Kunihiko SADAKANE. Variable-Order De Bruijn Graphs. *Proceedings of the 2015 Data Compression Conference*. IEEE Computer Society, 2015, p. 383-392. DOI : 10.1109/DCC.2015.70.

[11] Alexander BOWE, Taku ONODERA, Kunihiko SADAKANE et Tetsuo SHIBUYA. Succinct de Bruijn Graphs. *Algorithms in Bioinformatics : 12th International Workshop, WABI 2012*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 225-235. DOI : 10.1007/978-3-642-33122-0_18.

[12] Coen BRON et Joep KERBOSCH. Algorithm 457 : Finding All Cliques of an Undirected Graph. *Communications of the ACM* 16.9 (1973), p. 575-577. DOI : 10.1145/362342.362367.

[13] Nicolaas Govert de BRUIJN. A combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam* 49(1946).7 (1946), p. 758-764.

[14] M BURROWS et Dj WHEELER. A block-sorting lossless data compression algorithm. *Research report* (1994).

[15] Ségolène CABOCHE, Christophe AUDEBERT, Yves LEMOINE et David HOT. Comparison of mapping algorithms used in high-throughput sequencing : Application to Ion Torrent data. *BMC Genomics* 15.1 (2014), p. 1-16. DOI : 10.1186/1471-2164-15-264.

[16] Mark J CHAISSON et Glenn TESLER. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR) : application and theory. *BMC Bioinformatics* 13 (2012), p. 238. DOI : 10.1186/1471-2105-13-238.

[17] Rayan CHIKHI et Guillaume RIZK. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology* 2 (2013), p. 1-9. DOI : 10.1186/1748-7188-8-22.

[18] Chen Shan CHIN, Paul PELUSO, Fritz J. SEDLAZECK, Maria NATTESTAD, Gregory T. CONCEPCION et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature Methods* 13.12 (2016), p. 1050-1054. DOI : 10.1038/nmeth.4035.

[19] Chen-Shan CHIN, David H ALEXANDER, Patrick MARKS, Aaron A KLAMMER, James DRAKE et al. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature Methods* 10 (2013), p. 563-569. DOI : 10.1038/nmeth.2474.

[20] Olivia CHOUDHURY, Ankush CHAKRABARTY et Scott J. EMRICH. HECIL : A hybrid error correction algorithm for long reads with iterative learning. *Scientific Reports* 8.1 (2018), p. 1-9. DOI : 10.1038/s41598-018-28364-3.

[21] Peter J A COCK, Christopher J. FIELDS, Naohisa GOTO, Michael L. HEUER et Peter M. RICE. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research* 38.6 (2009), p. 1767-1771. DOI : 10.1093/nar/gkp1137.

[22] Matei DAVID, Misko DZAMBA, Dan LISTER, Lucian ILIE et Michael BRUDNO. SHRiMP2 : Sensitive yet practical short read mapping. *Bioinformatics* 27.7 (2011), p. 1011-1012. DOI : 10.1093/bioinformatics/btr046.

[23] E. W. DIJKSTRA. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik* 1.1 (1959), p. 269-271. DOI : 10.1007/BF01386390.

[24] David EPPSTEIN, Maarten LÖFFLER et Darren STRASH. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. *Algorithms and Computation*. Springer Berlin Heidelberg, 2010, p. 403-414. DOI : 10.1007/978-3-642-17517-6_36.

[25] David EPPSTEIN et Darren STRASH. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *Experimental Algorithms*. Springer Berlin Heidelberg, 2011, p. 364-375. DOI : 10.1007/978-3-642-20662-7_31.

[26] B EWING, L D HILLIER et M C WENDL. Base-Calling of Automated Sequencer Traces Using Phred. I. Accuracy Assessment. *Genome Research* 8.206 (1998), p. 186-194. DOI : 10.1101/gr.8.3.175.

[27] B EWING, L D HILLIER et M C WENDL. Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities. *Genome Research* 8.206 (1998), p. 186-194. DOI : 10.1101/gr.8.3.186.

[28] P FERRAGINA et G MANZINI. Opportunistic data structures with applications. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science* FOCS '00 (2000), p. 390-398. DOI : 10.1109/SFCS.2000.892127.

[29] Can FIRTINA, Ziv BAR-JOSEPH, Can ALKAN et A Ercument CICEK. Hercules : a profile HMM-based hybrid error correction algorithm for long reads. *Nucleic Acids Research* 46.21 (2018). DOI : 10.1093/nar/gky724.

[30] Robert W FLOYD. Algorithm 97 : Shortest Path. *Communications of the ACM* 5.6 (1962), p. 345-. DOI : 10.1145/367766.368168.

[31] M FRAZIER, R A GIBBS, D M MUZNY, S E SCHERER, J B BOUCK et al. Initial sequencing and analysis of the human genome. *Nature* 409.6822 (2001), p. 860-921. DOI : 10.1038/35057062.

[32] I J GOOD. Normal Recurring Decimals. *Journal of the London Mathematical Society* s1-21.3 (1946), p. 167-169. DOI : 10.1112/jlms/s1-21.3.167.

[33] Sara GOODWIN, James GURTOWSKI, Scott ETHE-SAYERS, Panchajanya DESHPANDE, Michael C SCHATZ et al. Oxford Nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome. *Genome Research* 25 (2015), p. 1750-1756. DOI : 10.1101/gr.191395.115.

[34] Sara GOODWIN, John D. MCPHERSON et W. Richard MCCOMBIE. Coming of age : Ten years of next-generation sequencing technologies. *Nature Reviews Genetics* 17.6 (2016), p. 333-351. DOI : 10.1038/nrg.2016.49.

[35] Kazuyoshi GOTOH, Teruo YASUNAGA, Takamasa IMAI, Daisuke MOTOOKA, Kazutoshi YOSHITAKE et al. Performance comparison of second and third-generation sequencers using a bacterial genome with two chromosomes. *BMC Genomics* 15.1 (2014), p. 699. DOI : 10.1186/1471-2164-15-699.

[36] Catherine GRASSO et Christopher LEE. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics* 20.10 (2004), p. 1546-1556. DOI : 10.1093/bioinformatics/bth126.

[37] Miodrag GUŽVIĆ. The history of DNA sequencing. *Journal of Medical Biochemistry* 32.4 (2013), p. 301-312. DOI : 10.2478/jomb-2014-0004.

[38] Thomas HACKL, Rainer HEDRICH, Jörg SCHULTZ et Frank FÖRSTER. Proovread : Large-scale high-accuracy PacBio correction through iterative short read consensus. *Bioinformatics* 30.21 (2014), p. 3004-3011. DOI : 10.1093/bioinformatics/btu392.

[39] Ehsan HAGHSHENAS, Faraz HACH, S Cenk SAHINALP et Cedric CHAUVE. CoLoRMap : Correcting Long Reads by Mapping short reads. *Bioinformatics* 32 (2016), p. i545-i551. DOI : 10.1093/bioinformatics/btw463.

[40] Yun HEO. Improving quality of high-throughput sequencing reads. Thèse de doct. Sous la dir. de Deming Chen. University of Illinois at Urbana–Champaign, 2015.

[41] Yun HEO, Xiao Long WU, Deming CHEN, Jian MA et Wen Mei HWU. BLESS : Bloom filter-based error correction solution for high-throughput sequencing reads. *Bioinformatics* 30.10 (2014), p. 1354-1362. DOI : 10.1093/bioinformatics/btu030.

[42] Ruifeng HU, Guibo SUN et Xiaobo SUN. LSCplus : a fast solution for improving long read accuracy by short read alignment. *BMC Bioinformatics* 17.1 (2016), p. 451. DOI : 10.1186/s12859-016-1316-y.

[43] Weichun HUANG, Leping LI, Jason R MYERS et Gabor T MARTH. ART : a next-generation sequencing read simulator. *Bioinformatics* 28 (2012), p. 593-594. DOI : 10.1093/bioinformatics/btr708.

[44] Lucian ILIE, Farideh FAZAYELI et Silvana ILIE. HiTEC : Accurate error correction in high-throughput sequencing data. *Bioinformatics* 27.3 (2011), p. 295-302. DOI : 10.1093/bioinformatics/btq653.

[45] Shaun D JACKMAN, Benjamin P VANDERVALK, Hamid MOHAMADI, Justin CHU, Sarah YEO et al. ABySS 2 . 0 : Resource-Efficient Assembly of Large Genomes using a Bloom Filter Effect of Bloom Filter False Positive Rate. *Genome Research* 27 (2017), p. 768-777. DOI : 10.1101/gr.214346.116.

[46] Miten JAIN, Sergey KOREN, Karen H MIGA, Josh QUICK, Arthur C RAND et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology* 36.4 (2018), p. 338. DOI : 10.1038/nbt.4060..

[47] S. J.M. JONES, I. BIROL, J. T. SIMPSON, K. WONG, S. D. JACKMAN et al. ABySS : A parallel assembler for short read sequence data. *Genome Research* 19.6 (2009), p. 1117-1123. DOI : 10.1101/gr.089532.108.

[48] Wei Chun KAO, Andrew H. CHAN et Yun S. SONG. ECHO : A reference-free short-read error correction algorithm. *Genome Research* 21.7 (2011), p. 1181-1192. DOI : 10.1101/gr.111351.110.

[49] Mehdi KCHOUK et Mourad ELLOUMI. An Error Correction and De-Novo Assembly Approach for Nanopore Reads Using Short Reads. *Current Bioinformatics* 13.3 (2018), p. 241-252. DOI : 10.2174/1574893612666170530073736.

[50] Mehdi KCHOUK, Jean Francois GIBRAT et Mourad ELLOUMI. Generations of Sequencing Technologies : From First to Next Generation. *Biology and Medicine* 09.03 (2017). DOI : 10.4172/0974-8369.1000395.

[51] David R KELLEY, Michael C SCHATZ et Steven L SALZBERG. Quake : quality-aware detection and correction of sequencing errors. *Genome Biology* 11.11 (2010), R116. DOI : 10.1186/gb-2010-11-11-r116.

[52] W James KENT. BLAT - The BLAST-Like Alignment Tool. *Genome Research* 12 (2002), p. 656-664. DOI : 10.1101/gr.229202..

[53] Szymon M KIELBASA, Raymond WAN, Kengo SATO, Szymon M KIEBASA, Paul HORTON et al. Adaptive seeds tame genomic sequence comparison. *Genome Research* 21 (2011), p. 487-493. DOI : 10.1101/gr.113985.110.

[54] Marek KOKOT, Maciej DŁUGOSZ et Sebastian DEOROWICZ. KMC3 : counting and manipulating k-mer statistics. *Bioinformatics* 33 (2017), p. 2759-2791. DOI : 10.1093/bioinformatics/btx304.

[55] Sergey KOREN, Gregory P HARHAY, Timothy P L SMITH, James L BONO, Dayna M HARHAY et al. Reducing assembly complexity of microbial genomes with single-molecule sequencing. *Genome Biology* 14.9 (sept. 2013), R101. DOI : 10.1186/gb-2013-14-9-r101.

[56] Sergey KOREN, Michael C SCHATZ, Brian P WALENZ, Jeffrey MARTIN, Jason T HOWARD et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology* 30.7 (2012), p. 693-700. DOI : 10.1038/nbt.2280.

[57] Sergey KOREN, Brian P WALENZ, Konstantin BERLIN, Jason R MILLER, Nicholas H BERGMAN et al. Canu : scalable and accurate long-read assembly via adaptive k -mer weighting and repeat separation. *Genome Research* 27 (2017), p. 722-736. DOI : 10.1101/gr.215087.116.

[58] Tomasz KOWALSKI, Szymon GRABOWSKI et Sebastian DEOROWICZ. Indexing Arbitrary-Length k-Mers in Sequencing Reads. *PLoS ONE* 10 (2015), p. 1-16. DOI : 10.1371/journal.pone.0133198.

[59] Stefan KURTZ. Reducing the space requirement of suffix trees. *Software : Practice and Experience* 29.13 (1999), p. 1149-1171. DOI : 10.1002/(SICI)1097-024X(199911)29:13<1149::AID-SPE274>3.0.CO;2-O.

[60] Stefan KURTZ, Adam PHILLIPPY, Arthur L DELCHER, Michael SMOOT, Martin SHUMWAY et al. Versatile and open software for comparing large genomes. *Genome Biology* 5 (jan. 2004), R12. DOI : 10.1186/gb-2004-5-2-r12.

[61] Sean LA, Ehsan HAGHSHENAS et Cedric CHAUVE. LRCstats, a tool for evaluating long reads correction methods. *Bioinformatics* 33 (2017), p. 3652-3654. DOI : 10.1093/bioinformatics/btx489.

[62] B LANGMEAD, C TRAPNELL, M POP et S L SALZBERG. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* 10.3 (2009), R25. DOI : 10.1186/gb-2009-10-3-r25.

[63] Ben LANGMEAD et Steven L SALZBERG. Fast gapped-read alignment with Bowtie 2. *Nat Methods* 9.4 (2012), p. 357-359. DOI : 10.1038/nmeth.1923.

[64] Christopher LEE. Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics* 19.8 (2003), p. 999-1008. DOI : 10.1093/bioinformatics/btg109.

[65] Christopher LEE, Catherine GRASSO et Mark F SHARLOW. Multiple sequence alignment using partial order graphs. *Bioinformatics* 18.3 (2002), p. 452-464. DOI : 10.1093/bioinformatics/18.3.452.

[66] Hayan LEE, James GURTOWSKI, Shinjae YOO, Shoshana MARCUS, W. Richard MCCOMBIE et al. Error correction and assembly complexity of single molecule sequencing reads. *bioRxiv* (2014), p. 006395. DOI : 10.1101/006395.

[67] Heng LI. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv* (2013). URL : https://arxiv.org/abs/1303.3997.

[68] Heng LI. Minimap and miniasm : Fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* 32.14 (2016), p. 2103-2110. DOI : 10.1093/bioinformatics/btw152.

[69] Heng LI. Minimap2 : pairwise alignment for nucleotide sequences. *Bioinformatics* 34.18 (2018), p. 3094-3100. DOI : 10.1093/bioinformatics/bty191.

[70] Heng LI et Richard DURBIN. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics* 26 (2010), p. 589-595. DOI : 10.1093/bioinformatics/btp698.

[71] Heng LI et Richard DURBIN. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 25.14 (2009), p. 1754-1760. DOI : 10.1093/bioinformatics/btp324.

[72] Ruiqiang LI, Hongmei ZHU, Jue RUAN, Wubin QIAN, Xiaodong FANG et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research* 20.2 (2010), p. 265-272. DOI : 10.1101/gr.097261.109.

[73] Yu LI, Renmin HAN, Chongwei BI, Mo LI, Sheng WANG et al. DeepSimulator : A deep simulator for Nanopore sequencing. *Bioinformatics* 34.17 (2018), p. 2899-2908. DOI : 10.1093/bioinformatics/bty223.

[74] Leandro LIMA, Camille MARCHET, Ségolène CABOCHE, Corinne DA SILVA, Benjamin ISTACE et al. Comparative assessment of long-read error-correction software applied to RNA-sequencing data. *bioRxiv* (2019). DOI : 10.1101/476622.

[75] Yu LIN et Pavel A PEVZNER. Manifold de Bruijn Graphs. *Algorithms in Bioinformatics : 14th International Workshop, WABI 2014*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014, p. 296-310. DOI : 10.1007/978-3-662-44753-6_22.

[76] Yongchao LIU, Jan SCHRÖDER et Bertil SCHMIDT. Musket : A multistage k-mer spectrum-based error corrector for Illumina sequence data. *Bioinformatics* 29.3 (2013), p. 308-315. DOI : 10.1093/bioinformatics/bts690.

[77] Thomas L MADDEN, Christiam CAMACHO, Ning MA, George COULOURIS, Vahram AVAGYAN et al. BLAST+ : architecture and applications. *BMC Bioinformatics* 10.1 (2009), p. 421. DOI : 10.1186/1471-2105-10-421.

[78] Mohammed-Amin MADOUI, Stefan ENGELEN, Corinne CRUAUD, Caroline BELSER, Laurie BERTRAND et al. Genome assembly using Nanopore-guided long and error-free DNA reads. *BMC Genomics* 16 (2015), p. 327. DOI : 10.1186/s12864-015-1519-z.

[79] Nicolas MAILLET, Guillaume COLLET, Thomas VANNIER, Dominique LAVENIER et Pierre PETERLONGO. Commet : Comparing and combining multiple metagenomic datasets. *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Belfast, United Kingdom, 2014. DOI : 10.1109/BIBM.2014.6999135.

[80] Udi MANBER et Gene MYERS. Suffix Arrays : A New Method for On-line String Searches. *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* SODA '90 (1990), p. 319-327. DOI : 10.1137/0222058.

[81] G MARCAIS et C KINGSFORD. Jellyfish : A fast k-mer counter. 1 (2012), p. 1-8.

[82] Guillaume MARÇAIS, James A YORKE et Aleksey ZIMIN. QuorUM : An Error Corrector for Illumina Reads. *PLoS ONE* 10 (2015), p. 1-13. DOI : 10.1371/journal.pone.0130821.

[83] Camille MARCHET. From reads to transcripts : de novo methods for the analysis of transcriptome second and third generation sequencing. Thèse de doct. Sous la dir. de Pierre Peterlongo. Université de Rennes 1, 2018.

[84] Pierre MARIJON, Rayan CHIKHI et Jean-Stéphane VARRÉ. yacrd and fpa : upstream tools for long-read genome assembly. *bioRxiv* (2019). DOI : 10.1101/674036.

[85] Walter Gilbert MAXAM et Allan M. A new method for sequencing DNA. 1977. *Proceedings of The National Academy of Sciences of The United States Of America* 74.2 (1977), p. 99-103. DOI : 10.1073/pnas.74.2.560.

[86] Giles MICLOTTE, Mahdi HEYDARI, Piet DEMEESTER, Stephane ROMBAUTS, Yves VAN DE PEER et al. Jabba : hybrid error correction for long sequencing reads. *Algorithms for Molecular Biology* 11 (2016), p. 10. DOI : 10.1186/s13015-016-0075-7.

[87] Alla MIKHEENKO, Andrey PRJIBELSKI, Dmitry ANTIPOV, Vladislav SAVELIEV et Alexey GUREVICH. Versatile genome assembly evaluation with QUAST-LG. *Bioinformatics* 34.13 (2018), p. i142-i150. DOI : 10.1093/bioinformatics/bty266.

[88] John C. MU, Hui JIANG, Amirhossein KIANI, Marghoob MOHIYUDDIN, Narges Bani ASADI et al. Fast and accurate read alignment for resequencing. *Bioinformatics* 28.18 (2012), p. 2366-2373. DOI : 10.1093/bioinformatics/bts450.

[89] Eugene W. MYERS, Granger G. SUTTON, Art L. DELCHER, Ian M. DEW, Dan P. FASULO et al. A whole-genome assembly of Drosophila. *Science* 287.5461 (2000), p. 2196-2204. DOI : 10.1126/science.287.5461.2196.

[90] Gene MYERS. Efficient Local Alignment Discovery amongst Noisy Long Reads. *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2014, p. 52-67. DOI : 10.1007/978-3-662-44753-6_5.

[91] Saul B NEEDLEMAN et Christian D WUNSCH. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48.3 (1970), p. 443-453. DOI : 10.1016/0022-2836(70)90057-4.

[92] Mihai POP, Adam PHILLIPPY, Arthur L. DELCHER et Steven L. SALZBERG. Comparative genome assembly. *Briefings in Bioinformatics* 5.3 (2004), p. 237-248. DOI : 10.1093/bib/5.3.237.

[93] Jason A REUTER, Damek V SPACEK et Michael P SNYDER. High-throughput sequencing technologies. *Molecular cell* 58 4 (2015), p. 586-597. DOI : 10.1016/j.molcel.2015.05.004.

[94] Daniel C. RICHTER, Felix OTT, Alexander F. AUCH, Ramona SCHMID et Daniel H. HUSON. MetaSim : A Sequencing Simulator for Genomics and Metagenomics. *Handbook of Molecular Microbial Ecology I : Metagenomics and Complementary Approaches* 3.10 (2011), p. 417-421. DOI : 10.1002/9781118010518.ch48.

[95] Guillaume RIZK, Dominique LAVENIER et Rayan CHIKHI. DSK : K-mer counting with very low memory usage. *Bioinformatics* 29.5 (2013), p. 652-653. DOI : 10.1093/bioinformatics/btt020.

[96] Camille Flye SAINTE-MARIE. Question 48. *L'Intermédiaire des Mathématiciens, vol. 1*. 1894, p. 107-110.

[97] Leena SALMELA. Correction of sequencing errors in a mixed set of reads. *Bioinformatics* 26.10 (2010), p. 1284-1290. DOI : 10.1093/bioinformatics/btq151.

[98] Leena SALMELA et Eric RIVALS. LoRDEC : Accurate and efficient long read error correction. *Bioinformatics* 30 (2014), p. 3506-3514. DOI : 10.1093/bioinformatics/btu538.

[99] Leena SALMELA et Jan SCHRÖDER. Correcting errors in short reads by multiple alignments. *Bioinformatics* 27.11 (2011), p. 1455-1461. DOI : 10.1093/bioinformatics/btr170.

[100] Leena SALMELA, Riku WALVE, Eric RIVALS et Esko UKKONEN. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics* 33 (2017), p. 799-806. DOI : 10.1093/bioinformatics/btw321.

[101] F SANGER, S NICKLEN et A R COULSON. DNA sequencing with chain-terminating inhibitors. *Proceedings of The National Academy of Sciences of The United States Of America* 74.12 (1977), p. 5463-5467. DOI : 10.1073/pnas.74.12.5463.

[102] C SCHENSTED. Longest Increasing and Decreasing Subsequences. *Canadian Journal of Mathematics* 13 (1961), p. 179-191. DOI : 10.4153/CJM-1961-015-3.

[103] Jan SCHRÖDER, Heiko SCHRÖDER, Simon J PUGLISI, Ranjan SINHA et Bertil SCHMIDT. SHREC : A short-read error correction method. *Bioinformatics* 25.17 (2009), p. 2157-2163. DOI : 10.1093/bioinformatics/btp379.

[104] Fritz J SEDLAZECK, Hayan LEE, Charlotte A. DARBY et Michael C. SCHATZ. Piercing the dark matter : bioinformatics of long-range sequencing and mapping. *Nature Reviews Genetics* 19 (2018), p. 329-346. DOI : 10.1038/s41576-018-0003-4.

[105] Fritz J SEDLAZECK, Philipp RESCHENEDER, Moritz SMOLKA, Han FANG, Maria NATTESTAD et al. Accurate detection of complex structural variations using single-molecule sequencing. *Nature Methods* 15.6 (2018), p. 461-468. DOI : 10.1038/s41592-018-0001-7.

[106] Jay SHENDURE, Shankar BALASUBRAMANIAN, George M. CHURCH, Walter GILBERT, Jane ROGERS et al. DNA sequencing at 40 : Past, present and future. *Nature* 550.7676 (2017), p. 345-353. DOI : 10.1038/nature24286.

[107] Jay SHENDURE et Hanlee JI. Next-generation DNA sequencing. *Nature Biotechnology* 26.10 (2008), p. 1135-1145. DOI : 10.1038/nbt1486.

[108] T F SMITH et M S WATERMAN. Identification of common molecular subsequences. *Journal of Molecular Biology* 147.1 (1981), p. 195-197. DOI : 10.1016/0022-2836(81)90087-5.

[109] Bianca K. STÖCKER, Johannes KÖSTER et Sven RAHMANN. SimLoRD : Simulation of Long Read Data. *Bioinformatics*. T. 32. 2016, p. 2704-2706. DOI : 10.1093/bioinformatics/btw286.

[110] German TISCHLER et Eugene W MYERS. Non Hybrid Long Read Consensus Using Local De Bruijn Graph Assembly. *bioRxiv* (2017). DOI : 10.1101/106252.

[111] Robert VASER, Ivan SOVIĆ, Niranjan NAGARAJAN et Mile ŠIKIĆ. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research* 27.5 (2017), p. 737-746. DOI : 10.1101/gr.214270.116.5.

[112] A VITERBI. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13.2 (1967), p. 260-269. DOI : 10.1109/TIT.1967.1054010.

[113] Greg A. VOLKMER, Gerard P. IRZYK, Xavier V. GOMES, Vinod B. MAKHIJANI, George T. ROTH et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature* 437.7057 (2005), p. 376-380. DOI : 10.1038/nature03959.

[114] Michaël VYVERMAN, Bernard DE BAETS, Veerle FACK et Peter DAWYNDT. EssaMEM : Finding maximal exact matches using enhanced sparse suffix arrays. *Bioinformatics* 29.6 (2013), p. 802-804. DOI : 10.1093/bioinformatics/btt042.

[115] Jeremy R. WANG, James HOLT, Leonard MCMILLAN et Corbin D. JONES. FMLRC : Hybrid long read error correction using an FM-index. *BMC Bioinformatics* 19.1 (2018), p. 1-11. DOI : 10.1186/s12859-018-2051-3.

[116] Stephen WARSHALL. A Theorem on Boolean Matrices. *Journal of the ACM* 9.1 (1962), p. 11-12. DOI : 10.1145/321105.321107.

[117] James Dewey WATSON et Francis Harry Compton CRICK. Molecular Structure of Nucleic Acids : A Structure for Deoxyribose Nucleic Acid. *Nature* 171 (1953), p. 737-738. DOI : 10.1038/171737a0.

[118] Ze Gang WEI et Shao Wu ZHANG. NPBSS : A new PacBio sequencing simulator for generating the continuous long reads with an empirical model. *BMC Bioinformatics* 19.1 (2018), p. 1-9. DOI : 10.1186/s12859-018-2208-0.

[119] Peter WEINER. Linear pattern matching algorithms. *Switching and Automata Theory, 1973. SWAT '08. IEEE Conference Record of 14th Annual Symposium on* (1973), p. 1-11. DOI : 10.1109/SWAT.1973.13.

[120] Wikimedia. URL : https://commons.wikimedia.org/wiki/File:Difference_DNA_RNA-EN.svg.

[121] Wikimedia. URL : https://commons.wikimedia.org/wiki/File:Didesoxy-Methode.svg.

[122] Wikimedia. URL : https://commons.wikimedia.org/wiki/File:Maxam-Gilbert_sequencing_en.svg.

[123] A. WYSOKER, T. FENNELL, G. MARTH, G. ABECASIS, J. RUAN et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* 25.16 (2009), p. 2078-2079. DOI : 10.1093/bioinformatics/btp352.

[124] Chuan Le XIAO, Ying CHEN, Shang Qian XIE, Kai Ning CHEN, Yan WANG et al. MECAT : Fast mapping, error correction, and de novo assembly for single-molecule sequencing reads. *Nature Methods* 14.11 (2017), p. 1072-1074. DOI : 10.1038/nmeth.4432.

[125] Chen YANG, Justin CHU, René L. WARREN et Inanç BIROL. NanoSim : Nanopore sequence read simulator based on statistical characterization. 2017. DOI : 10.1093/gigascience/gix010.

[126] Xiao YANG, Sriram P. CHOCKALINGAM et Srinivas ALURU. A survey of error-correction methods for next-generation sequencing. *Briefings in Bioinformatics* 14.1 (2013), p. 56-66. DOI : 10.1093/bib/bbs015.

[127] Chengxi YE et Zhanshan (Sam) MA. Sparc : a sparsity-based consensus algorithm for long erroneous sequencing reads. *PeerJ* 4 (2016). DOI : 10.7717/peerj.2016.

[128] Xin YIN, Zhao SONG, Karin DORMAN et Aditya RAMAMOORTHY. PREMIER Turbo : Probabilistic error-correction using Markov inference in errored reads using the turbo principle. *2013 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2013 - Proceedings* (2013), p. 73-76. DOI : 10.1109/GlobalSIP.2013.6736816.

[129] Mengyao ZHAO, Wan Ping LEE, Erik P. GARRISON et Gabor T. MARTH. SSW Library : An SIMD Smith-Waterman C/C++ library for use in genomic applications. *PLoS ONE* 8.12 (2013), p. 1-7. DOI : 10.1371/journal.pone.0082138.

Correction de données de séquençage de troisième génération

Les objectifs de cette thèse s'inscrivent dans la large problématique du traitement des données issues de séquenceurs à très haut débit, et plus particulièrement des *reads* longs, issus de séquenceurs de troisième génération. Les aspects abordés dans cette problématiques se concentrent principalement sur la correction des erreurs de séquençage, et sur l'impact de la correction sur la qualité des analyses sous-jacentes, plus particulièrement sur l'assemblage.

Dans un premier temps, l'un des objectifs de cette thèse est de permettre d'évaluer et de comparer la qualité de la correction fournie par les différentes méthodes de correction hybride (utilisant des *reads* courts en complément) et d'auto-correction (se basant uniquement sur l'information contenue dans les *reads* longs) de l'état de l'art. Une telle évaluation permet d'identifier aisément quelle méthode de correction est la mieux adaptée à un cas donné, notamment en fonction de la complexité du génome étudié, de la profondeur de séquençage, ou du taux d'erreurs des *reads*. De plus, les développeurs peuvent ainsi identifier les limitations des méthodes existantes, afin de guider leurs travaux et de proposer de nouvelles solutions visant à pallier ces limitations. Un nouvel outil d'évaluation, proposant de nombreuses métriques supplémentaires par rapport au seul outil disponible jusqu'alors, a ainsi été développé. Cet outil, combinant une approche par alignement multiple à une stratégie de segmentation, permet également une réduction considérable du temps nécessaire à l'évaluation. À l'aide de cet outil, un *benchmark* de l'ensemble des méthodes de correction disponibles est présenté, sur une large variété de jeux de données, de profondeur de séquençage, de taux d'erreurs et de complexité variable, de la bactérie *A. baylyi* à l'humain. Ce *benchmark* a notamment permis d'identifier deux importantes limitations des outils existants : les *reads* affichant des taux d'erreurs supérieurs à 30%, et les *reads* de longueur supérieure à 50 000 paires de bases.

Le deuxième objectif de cette thèse est alors la correction des *reads* extrêmement bruités. Pour cela, un outil de correction hybride, combinant différentes approches de l'état de l'art, a été développé afin de surmonter les limitations des méthodes existantes. En particulier, cet outil combine une stratégie d'alignement des *reads* courts sur les *reads* longs à l'utilisation d'un graphe de de Bruijn, ayant la particularité d'être d'ordre variable. Le graphe est ainsi utilisé afin de relier les *reads* alignés, et donc de corriger les régions non couvertes des *reads* longs. Cette méthode permet ainsi de corriger des *reads* affichant des taux d'erreurs atteignant jusqu'à 44%, tout en permettant un meilleur passage à l'échelle sur de larges génomes et une diminution du temps de traitement, par rapport aux méthodes de l'état de l'art les plus efficaces.

Enfin, le troisième objectif de cette thèse est la correction des *reads* extrêmement longs. Pour cela, un outil utilisant cette fois une approche par auto-correction a été développé, en combinant, de nouveau, différentes méthodologies de l'état de l'art. Plus précisément, une stratégie de calcul des chevauchements entre les *reads*, puis une double étape de correction, par alignement multiple puis par utilisation de graphes de de Bruijn locaux, sont utilisées ici. Afin de permettre à cette méthode de passer efficacement à l'échelle sur les *reads* extrêmement longs, la stratégie de segmentation mentionnée précédemment a été généralisée. Cette méthode d'auto-correction permet ainsi de corriger des *reads* atteignant jusqu'à 340 000 paires de bases, tout en permettant un excellent passage à l'échelle sur des génomes plus complexes, tels que celui de l'humain.

Mots-clés : Séquençage à haut débit, Correction d'erreurs, Assemblage, Graphe de de Bruijn, Alignement multiple

Error correction of third-generation sequencing data

The aims of this thesis are part of the vast problematic of high-throughput sequencing data analysis. More specifically, this thesis deals with long reads from third-generation sequencing technologies. The aspects tackled in this topic mainly focus on error correction, and on its impact on downstream analyses such a *de novo* assembly.

As a first step, one of the objectives of this thesis is to evaluate and compare the quality of the error correction provided by the state-of-the-art tools, whether they employ a hybrid (using complementary short reads) or a self-correction (relying only on the information contained in the long reads sequences) strategy. Such an evaluation allows to easily identify which method is best tailored for a given case, according to the genome complexity, the sequencing depth, or the error rate of the reads. Moreover, developpers can thus identify the limiting factors of the existing methods, in order to guide their work and propose new solutions allowing to overcome these limitations. A new evaluation tool, providing a wide variety of metrics, compared to the only tool previously available, was thus developped. This tool combines a multiple sequence alignment approach and a segmentation strategy, thus allowing to drastically reduce the evaluation runtime. With the help of this tool, we present a benchmark of all the state-of-the-art error correction methods, on various datasets from several organisms, spanning from the *A. baylyi* bacteria to the human. This benchmark allowed to spot two major limiting factors of the existing tools: the reads displaying error rates above 30%, and the reads reaching more than 50 000 base pairs.

The second objective of this thesis is thus the error correction of highly noisy long reads. To this aim, a hybrid error correction tool, combining different strategies from the state-of-the-art, was developped, in order to overcome the limiting factors of existing methods. More precisely, this tool combines a short reads alignment strategy to the use of a variable-order de Bruijn graph. This graph is used in order to link the aligned short reads, and thus correct the uncovered regions of the long reads. This method allows to process reads displaying error rates as high as 44%, and scales better to larger genomes, while allowing to reduce the runtime of the error correction, compared to the most efficient state-of-the-art tools.

Finally, the third objectif of this thesis is the error correction of extremely long reads. To this aim, a self-correction tool was developed, by combining, once again, different methologies from the state-of-the-art. More precisely, an overlapping strategy, and a two phases error correction process, using multiple sequence alignement and local de Bruijn graphs, are used. In order to allow this method to scale to extremely long reads, the aforementioned segmentation strategy was generalized. This self-correction methods allows to process reads reaching up to 340 000 base pairs, and manages to scale very well to complex organisms such as the human genome.

Keywords: High-throughput sequencing, Error correction, Assembly, De Bruijn graphs, Multiple Sequence Alignment